

**GENETIC PROGRAMMING:  
PROGRAMMING COMPUTERS BY  
MEANS OF NATURAL SELECTION**

**WEDNESDAY – JUNE 4, 1997  
PASADENA**

**JOHN R. KOZA**

Consulting Professor  
Computer Science Department and Symbolic Systems Program  
258 Gates Building  
Stanford University  
Stanford, CA 94305 USA  
E-MAIL: `Koza@Cs.Stanford.Edu`  
PHONE: 415-941-0336  
FAX: 415-941-9430  
WWW: `http://www-cs-faculty.stanford.edu/~koza/`

# OUTLINE

- **Background on Genetic Algorithm**
- **Genetic Programming**
  - Example of GP
  - Automatically Defined Functions (ADFs)
  - Memory, State, Mental Models, Data structures
  - Iteration and recursion
  - Evolutionary Selection of Architecture
  - Evolution of Architecture using Architecture-Altering Operations
  - Implementation on parallel computer
  - Rapidly Reconfigurable field-programmable gate arrays (FPGAs) and evolvable hardware
  - Implementation of GP in assembly code
  - Cellular encoding (Developmental GP)
  - Selected problems where GP is competitive with human solutions
  - Automated design of analog circuits
  - Promising application areas for GP
  - Directions for possible future research
  - Bibliography, conferences, E-Mail lists, WWW and FTP sites

## THE GENETIC ALGORITHM (GA)

- The *genetic algorithm* is a mathematical algorithm that transforms a set (population) of mathematical objects (typically fixed-length binary character strings), each with an associated fitness value, into a new set (new generation of the population) of offspring objects, using operations patterned after naturally-occurring genetic operations and the Darwinian principle of reproduction and survival of the fittest.

### EXAMPLE (POPULATION OF 4 STRINGS OF LENGTH 3 OVER ALPHABET OF SIZE 2)

#### Generation 0

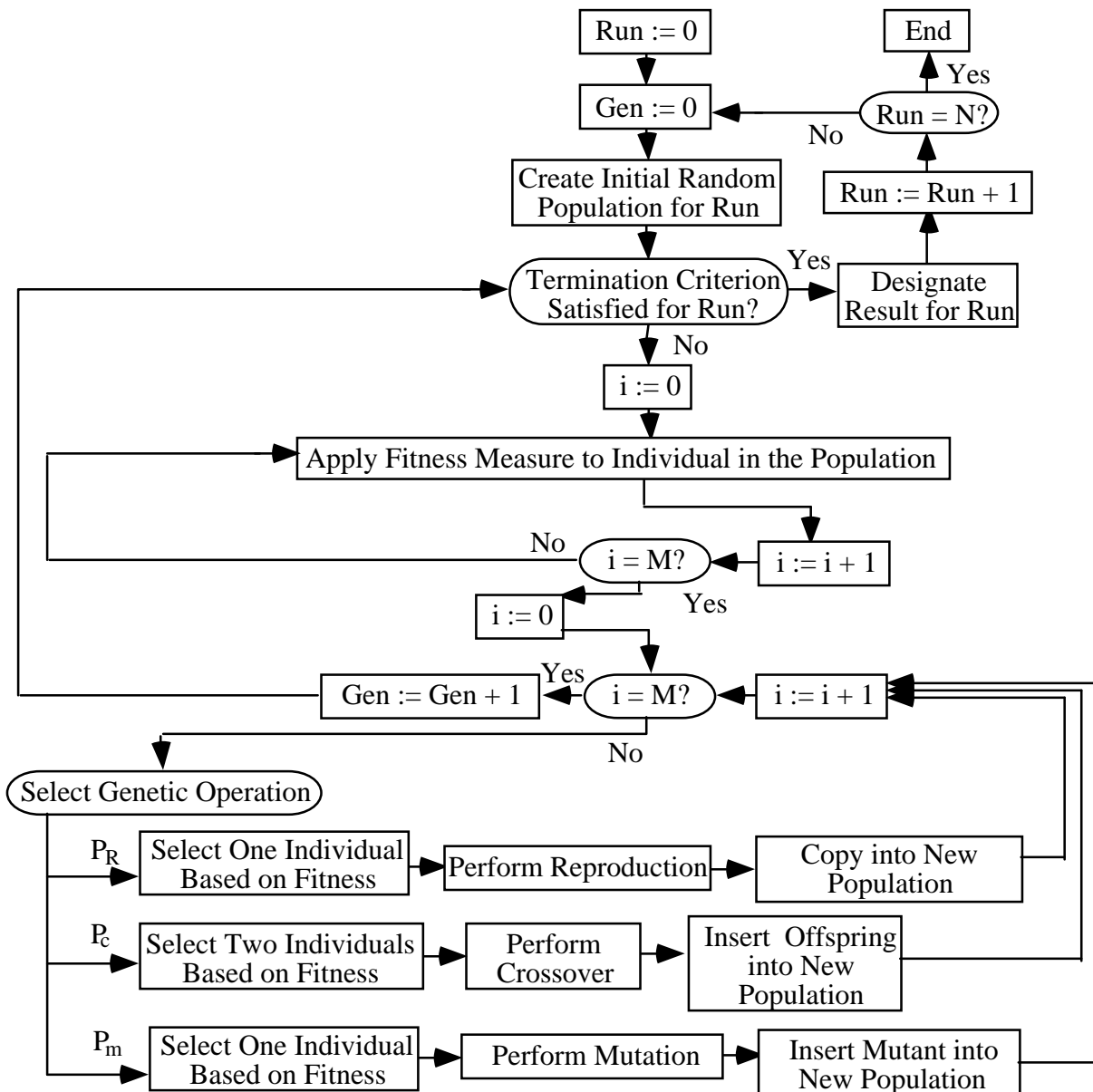
<u>Individuals</u> <u>in Population</u>	<u>Fitness</u> <u>Measure</u>
011	\$3

#### Generation 1

<u>Offspring</u> <u>Population</u>
111

<b>001</b>	<b>\$1</b>	<b>→</b>	<b>010</b>
<b>110</b>	<b>\$6</b>		<b>110</b>
<b>010</b>	<b>\$2</b>		<b>010</b>

# FLOWCHART FOR THE BASIC GENETIC ALGORITHM



# **PROBABILISTIC SELECTION**

## **INDIVIDUALS ARE SELECTED TO PARTICIPATE IN THE GENETIC OPERATIONS BASED ON FITNESS**

- **Better individuals are usually chosen**
- **The best individual is not necessarily chosen**
- **The worst individual is not necessarily excluded**
- **Thus, there is SOME greedy hill-climbing**
- **But, there is considerable selection of individuals that are the INFERIOR based on the current evidence of the search**
- **Resembles simulated annealing**
- **Unlike almost everything else**

# **THE PROBLEM OF AUTOMATIC PROGRAMMING (PROGRAM SYNTHESIS)**

**"How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?"**

**---Attributed to Arthur Samuel - about 1959**

# **AUTOMATIC PROGRAMMING**

**"WYWIWYG" – "WHAT YOU WANT IS  
WHAT YOU GET"**

- **Produces an entity that runs on a computer (i.e., a computer program or something that is easily convertible into a program)**
- **Requires a minimum of user-supplied information**
- **Solves a broad variety of problems**
- **Scalability to larger problems**
- **Implements all the familiar programming constructs – parameterizable subroutines, memory, iteration, recursion, data structures**
- **Doesn't require the user to prespecify the size and shape of the solution**
- **Doesn't require the user to identify subgoals, handcraft operators, decompose the problem, or intervene interactively during the run**
- **Produces some results that are competitive with human performance by programmers,**

**designers, or mathematicians or that are publishable in their own right**

- Is well-defined, has no hidden steps, and produces replicable results**

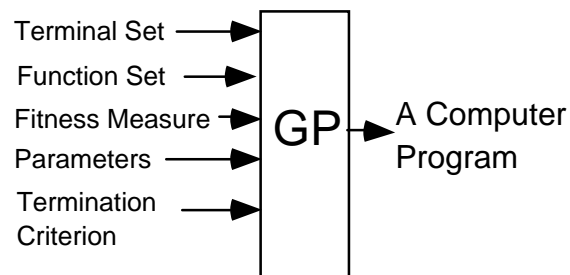
## **GENETIC PROGRAMMING (GP)**

**"Genetic programming *is* automatic programming. For the first time since the idea of automatic programming was first discussed in the late 40's and early 50's, we have a set of non-trivial, non-tailored, computer-generated programs that satisfy Samuel's exhortation: Tell the computer what to do, not how to do it.' "**

**– John Holland, University of Michigan, 1997**

# **FIVE MAJOR PREPARATORY STEPS FOR GP**

- **Determining the set of terminals**
- **Determining the set of functions**
- **Determining the fitness measure**
- **Determining the parameters**
  - population size
  - number of generations
  - other minor parameters
- **Determining the method for designating a result and the criterion for terminating a run**



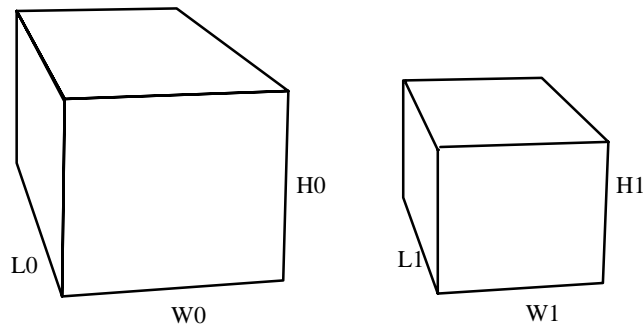
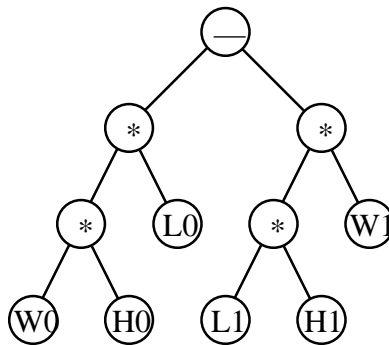
**10 FITNESS-CASES SHOWING THE  
VALUE OF THE DEPENDENT  
VARIABLE,  $D$ , ASSOCIATED WITH THE  
VALUES OF THE SIX INDEPENDENT  
VARIABLES,  $L_0, W_0, H_0, L_1, W_1, H_1$**

<b>Fitness case</b>	$L_0$	$W_0$	$H_0$	$L_1$	$W_1$	$H_1$	$D$
<b>1</b>	<b>3</b>	<b>4</b>	<b>7</b>	<b>2</b>	<b>5</b>	<b>3</b>	<b>54</b>
<b>2</b>	<b>7</b>	<b>10</b>	<b>9</b>	<b>10</b>	<b>3</b>	<b>1</b>	<b>600</b>
<b>3</b>	<b>10</b>	<b>9</b>	<b>4</b>	<b>8</b>	<b>1</b>	<b>6</b>	<b>312</b>
<b>4</b>	<b>3</b>	<b>9</b>	<b>5</b>	<b>1</b>	<b>6</b>	<b>4</b>	<b>111</b>
<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>7</b>	<b>6</b>	<b>1</b>	<b>-18</b>
<b>6</b>	<b>3</b>	<b>3</b>	<b>1</b>	<b>9</b>	<b>5</b>	<b>4</b>	<b>-171</b>
<b>7</b>	<b>5</b>	<b>9</b>	<b>9</b>	<b>1</b>	<b>7</b>	<b>6</b>	<b>363</b>
<b>8</b>	<b>1</b>	<b>2</b>	<b>9</b>	<b>3</b>	<b>9</b>	<b>2</b>	<b>-36</b>
<b>9</b>	<b>2</b>	<b>6</b>	<b>8</b>	<b>2</b>	<b>6</b>	<b>10</b>	<b>-24</b>
<b>10</b>	<b>8</b>	<b>1</b>	<b>10</b>	<b>7</b>	<b>5</b>	<b>1</b>	<b>45</b>

# SOLUTION USING GENETIC PROGRAMMING WITHOUT AUTOMATICALLY DEFINED FUNCTIONS (ADFs)

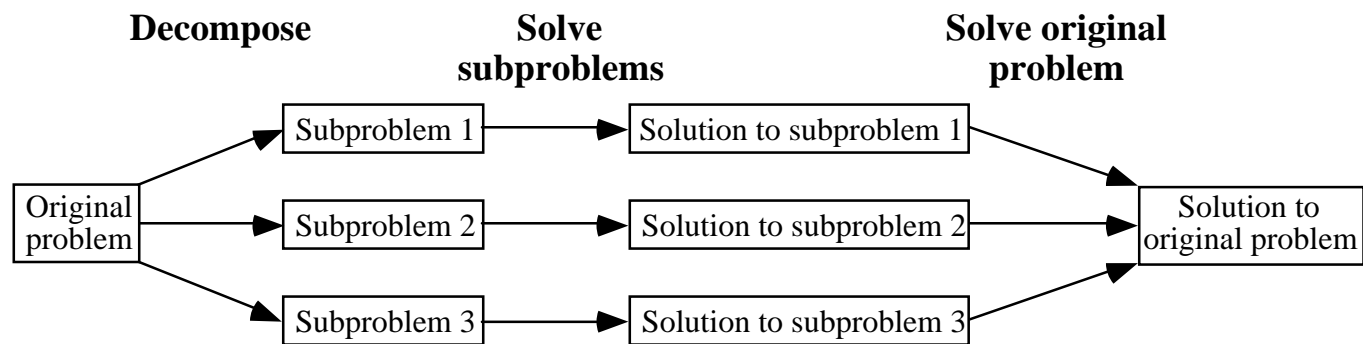
**( - ( \* ( \* W0 L0 ) H0 )  
 ( \* ( \* W1 L1 ) H1 ) )**

**D = W0\*L0\*H0 - W1\*L1\*H1**



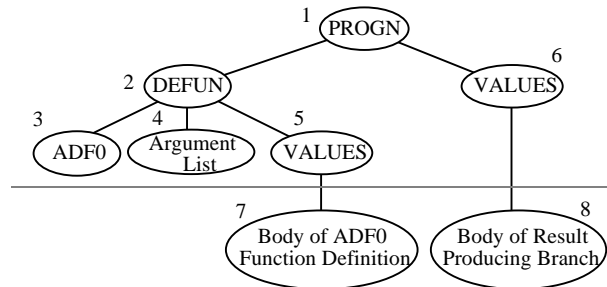
# **AUTOMATICALLY DEFINED FUNCTIONS**

## **TOP-DOWN VIEW OF THREE STEP HEIRARCHICAL PROBLEM-SOLVING PROCESS**



- **Decompose a problem into subproblems**
- **Solve the subproblems**
- **Assemble the solutions of the subproblems into a solution for the overall problem**

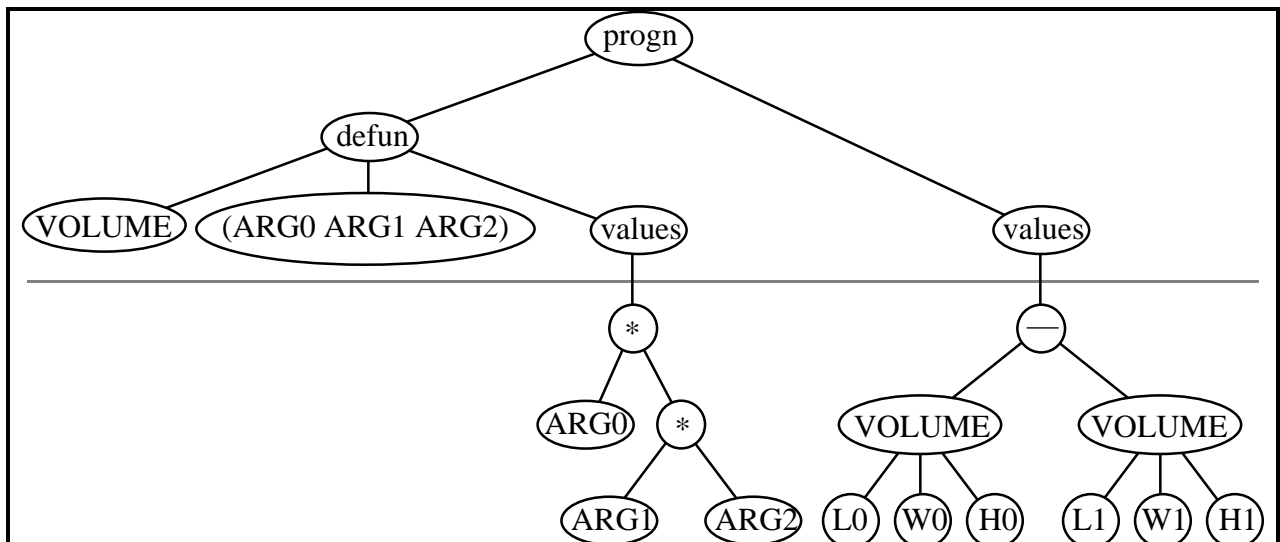
# AN OVERALL COMPUTER PROGRAM CONSISTING OF ONE FUNCTION- DEFINING BRANCH AND ONE RESULT- PRODUCING BRANCH



# 100%-CORRECT PROGRAM FOR THE TWO-BOXES PROBLEM with ADFS

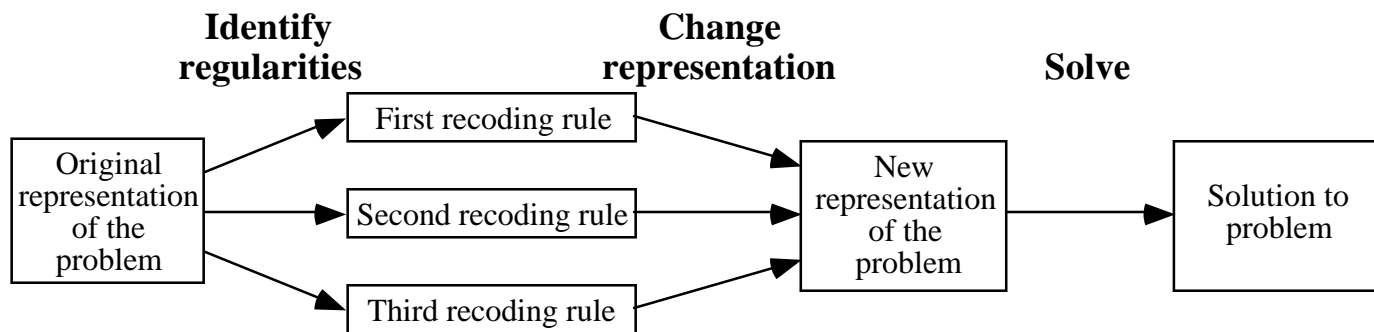
```
(progn
  (defun volume (arg0 arg1 arg2)
    (values
      (* arg0 (* arg1 arg2))))

  (values (- (volume L0 W0 H0)
             (volume L1 W1 H1))))
```



# **AUTOMATICALLY DEFINED FUNCTIONS**

## **BOTTOM-UP VIEW OF THREE STEP HEIRARCHICAL PROBLEM-SOLVING PROCESS**



- **Identify regularities**
- **Change the representation**
- **Solve the overall problem**

**AFTER THE CHANGE OF  
REPRESENTATION, THERE ARE TWO  
NEW VARIABLES –  $V_0$  AND  $V_1$**

<b>Fitness case</b>	$L_0$	$W_0$	$H_0$	$L_1$	$W_1$	$H_1$	$V_0$	$V_1$	$D$
<b>1</b>	<b>3</b>	<b>4</b>	<b>7</b>	<b>2</b>	<b>5</b>	<b>3</b>	<b>84</b>	<b>30</b>	<b>54</b>
<b>2</b>	<b>7</b>	<b>10</b>	<b>9</b>	<b>10</b>	<b>3</b>	<b>1</b>	<b>630</b>	<b>30</b>	<b>600</b>
<b>3</b>	<b>10</b>	<b>9</b>	<b>4</b>	<b>8</b>	<b>1</b>	<b>6</b>	<b>360</b>	<b>48</b>	<b>312</b>
<b>4</b>	<b>3</b>	<b>9</b>	<b>5</b>	<b>1</b>	<b>6</b>	<b>4</b>	<b>135</b>	<b>24</b>	<b>111</b>
<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>7</b>	<b>6</b>	<b>1</b>	<b>24</b>	<b>42</b>	<b>–18</b>
<b>6</b>	<b>3</b>	<b>3</b>	<b>1</b>	<b>9</b>	<b>5</b>	<b>4</b>	<b>9</b>	<b>180</b>	<b>–171</b>
<b>7</b>	<b>5</b>	<b>9</b>	<b>9</b>	<b>1</b>	<b>7</b>	<b>6</b>	<b>405</b>	<b>42</b>	<b>363</b>
<b>8</b>	<b>1</b>	<b>2</b>	<b>9</b>	<b>3</b>	<b>9</b>	<b>2</b>	<b>18</b>	<b>54</b>	<b>–36</b>
<b>9</b>	<b>2</b>	<b>6</b>	<b>8</b>	<b>2</b>	<b>6</b>	<b>10</b>	<b>96</b>	<b>120</b>	<b>–24</b>
<b>10</b>	<b>8</b>	<b>1</b>	<b>10</b>	<b>7</b>	<b>5</b>	<b>1</b>	<b>80</b>	<b>35</b>	<b>45</b>

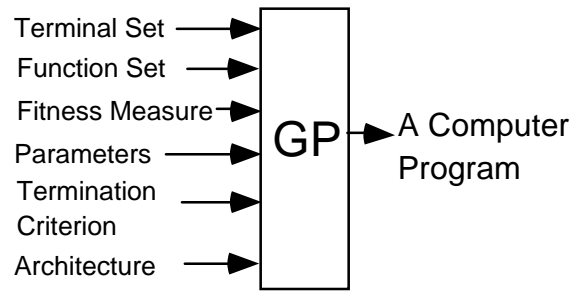
## **8 MAIN POINTS – AUTOMATICALLY DEFINED FUNCTIONS**

- **ADFs work.**
- **ADFs do not solve problems in the style of human programmers.**
- **ADFs reduce the computational effort required to solve a problem.**
- **ADFs usually improve the parsimony of the solutions to a problem.**
- **As the size of a problem is scaled up, the size of solutions increases more slowly with ADFs than without them.**
- **As the size of a problem is scaled up, the computational effort required to solve a problem increases more slowly with ADFs than without them.**
- **The advantages in terms of computational effort and parsimony conferred by ADFs increase as the size of the problem is scaled up.**

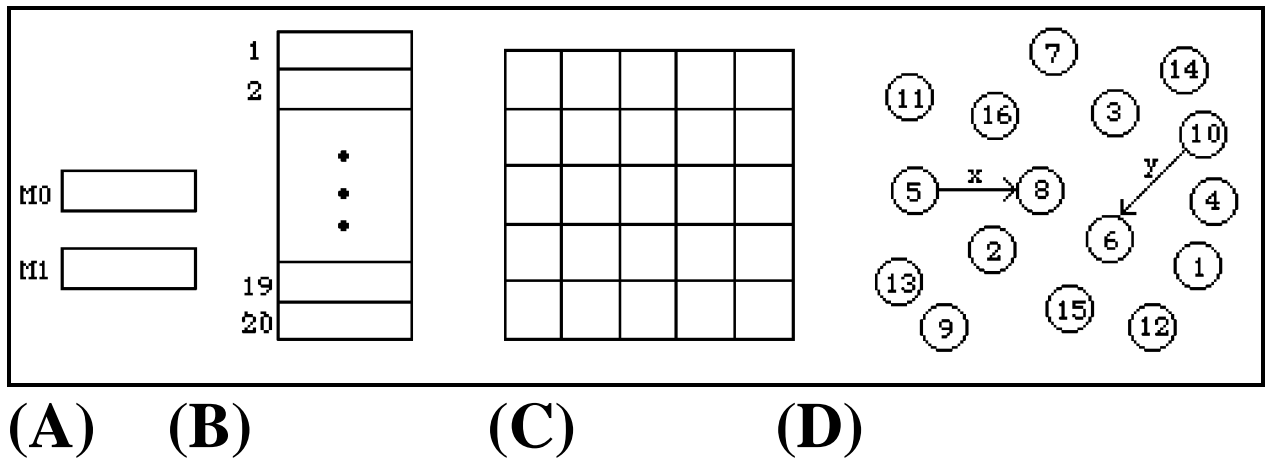
- **Genetic programming can evolve the architecture of the solution to a problem at the same time that it solves a problem.**

# **SIX MAJOR PREPARATORY STEPS FOR GENETIC PROGRAMMING WITH AUTOMATICALLY DEFINED FUNCTIONS (ADFs)**

- **determining the set of terminals**
- **determining the set of functions**
- **determining the fitness measure**
- **determining the parameters**
  - population size
  - number of generations
  - other minor parameters
- **determining the method for designating a result and the criterion for terminating a run**
- **determining the architecture of the overall program**
  - number of ADFs
  - number of arguments possessed by each ADF
  - hierarchical references, if any, among the ADFs



# APPROACHES TO MEMORY, STATE, AND MENTAL MODELS



- (A) Settable variables (*Genetic Programming*, Koza 1992) using terminals M0 and M1 and functions (SETM0 X) and (SETM1 Y)
- (B) Indexed memory similar to linear computer memory (Teller 1994) using (READ K) and (WRITE X K)
- (C) Memory isomorphic to world (Andre 1994)
- (D) Point-labeled, line-labeled directed graphs (Brave 1995, 1996)

## LANGDON'S DATA STRUCTURES

- Stacks

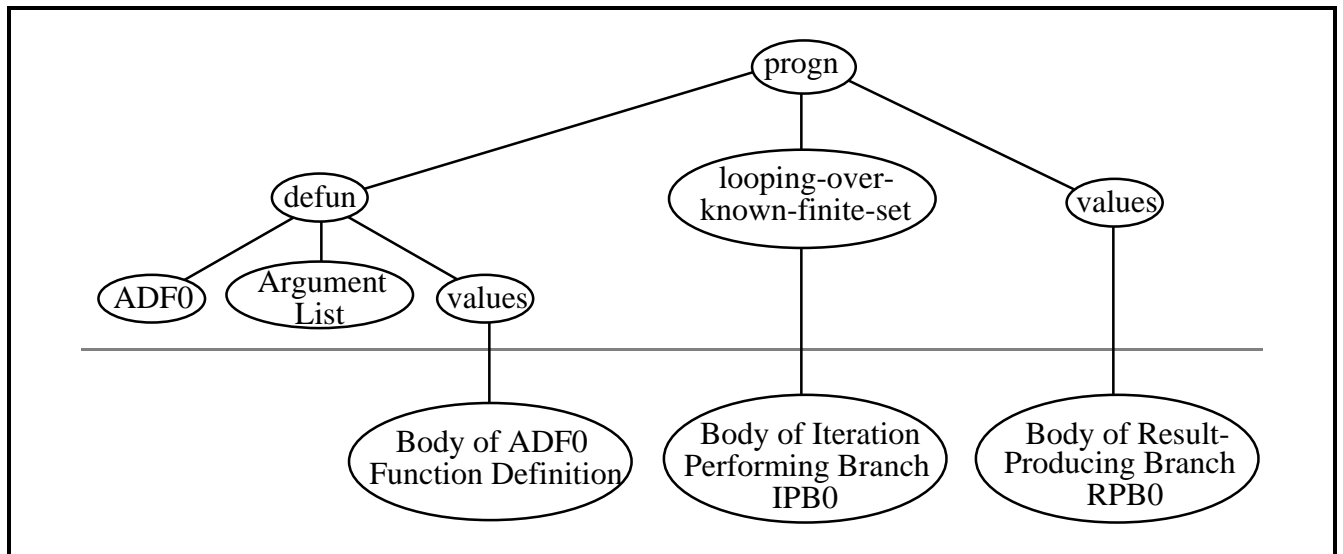
- **Queues**
- **Lists**
- **Rings**

# ITERATION

- **Four parts**
  - initialization branch
  - termination branch
  - work-performing branch
  - update branch
- **Problems**
  - nested iterations
  - unsatisfiable termination predicates
- **DU ("Do Until") operation (GP-1)**
  - number of steps in an iteration
  - number of iterations
- **Restricted iteration (GP-2)**
- **Rationing dynamically created iterations**
  - Restricted Iteration Creation (RIC)
  - Iteration Group Creation (IGC)

# RESTRICTED ITERATION

**Overall program consisting of an automatically defined function ADF0, an iteration-performing branch IPB0, and a result-producing branch RPB0.**



# **FINDING THE ARCHITECTURE OF THE AUTOMATICALLY DEFINED FUNCTIONS**

## **MANUAL METHODS**

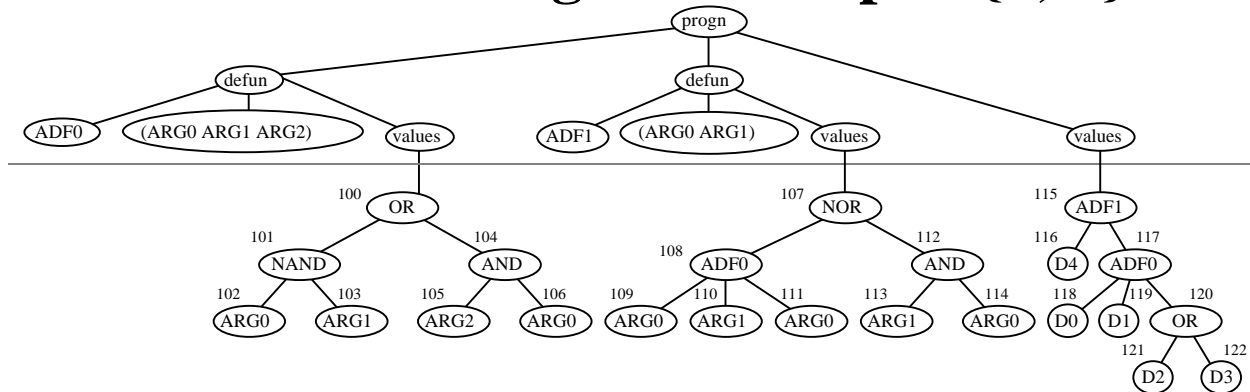
- **prospective analysis of the problem**
- **seemingly sufficient capacity (over-specification)**
- **affordable capacity**
- **retrospective analysis of the results of actual runs**

## **AUTOMATED METHODS**

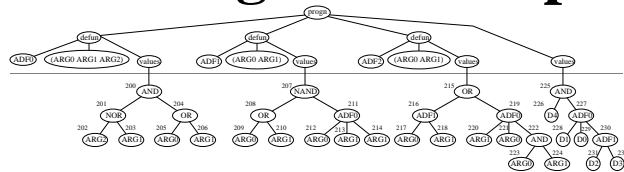
- **evolutionary selection of the architecture**
- **evolution of architecture using architecture-altering operations**

# POINT TYPING – STRUCTURE-PRESERVING CROSSOVER – ARCHITECTURALLY DIVERSE POPULATION

**Parent A with an argument map of {3, 2}**



**Parent B with an argument map of {3, 2, 2}**



**Parent C with an argument map of {4, 2}**



# **NEW ARCHITECTURE-ALTERING OPERATORS**

## **SPECIALIZATION – REFINEMENT – CASE SPLITTING**

- **Branch duplication**
- **Argument duplication**
- **Branch creation**
- **Argument creation**

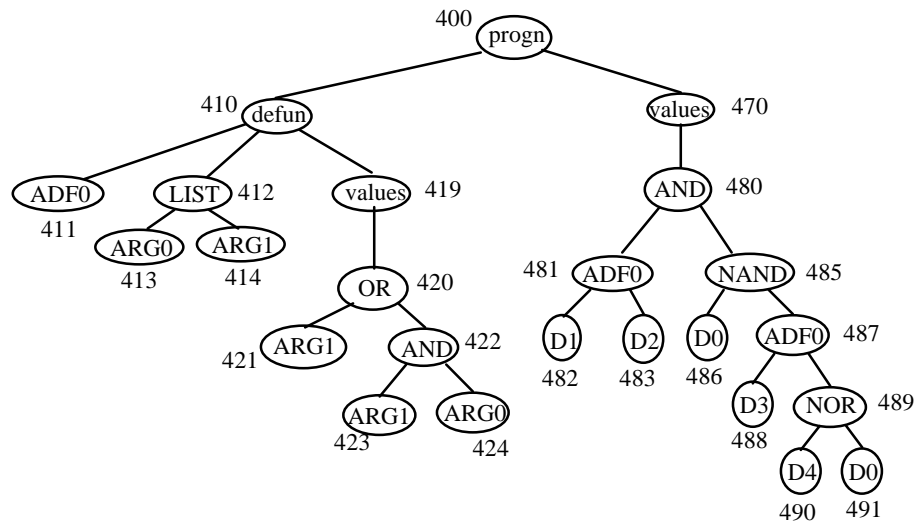
## **GENERALIZATION**

- **Branch deletion**
- **Argument deletion**

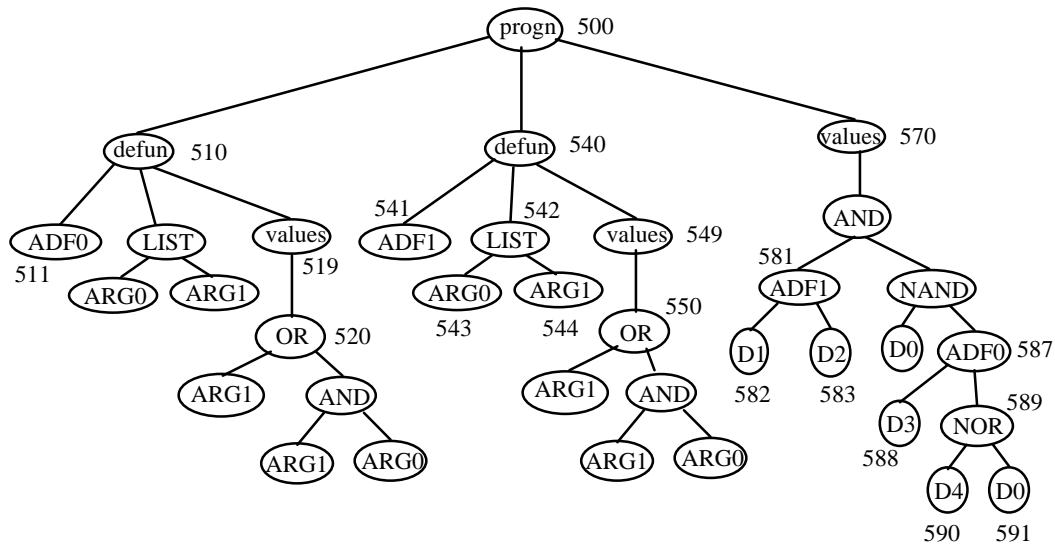
# PROTEIN ALIGNMENT OF "A" AND "B" PROTEINS

First.protein	MRIKFLVVLAVICLFAHYASASGMGGDKKP KDAPKPKDAP KPKEVKPVKA	50
Second.protein	MRIKFLVVLAVICLLAHYASASGMGGDKKP KDAPKPKDAP KPKEVKPVKA	50
First.protein	ESSEYEIEVI KHQKEKTEKK EKEKKIHVET KKEVKKKEKK QIPCSEKLLD	100
Second.protein	DSSEYEIEVI KHQKEKTEKK EKEKKAHVEI KKKIKNKEKK FVPCSEILKD	100
First.protein	EKLDCETKGV PAGYKAIFKF ITENEECDWT CDYEALPPPP GAKKDDKKEK	149
Second.protein	EKLECEKNAT P-GYKALFEF KESESFCWE CDYEAL---P GAKKDEKKEK	146
First.protein	KIVKVMKPPK EKPPKKLRKE CSGEKVIKFQ NCLVKIRGLI AFGDKTKNFD	199
Second.protein	KVVKVIKPPK EKPPKKPRKE CSGEKVIKFQ NCLVKIRGLI AFGDKTKNFD	196
First.protein	KKFAKLTVQ GKKGAKKAG GKKAAPKPGP KPGPK----Q ADKP-----	239
Second.protein	KKFAKLTVQ GKKGAKKAG GKKAEPKPGP KPAPKPGPKP APKPVPKPAD	246
First.protein	--KDAKK	244
Second.protein	KPKDAKK	253

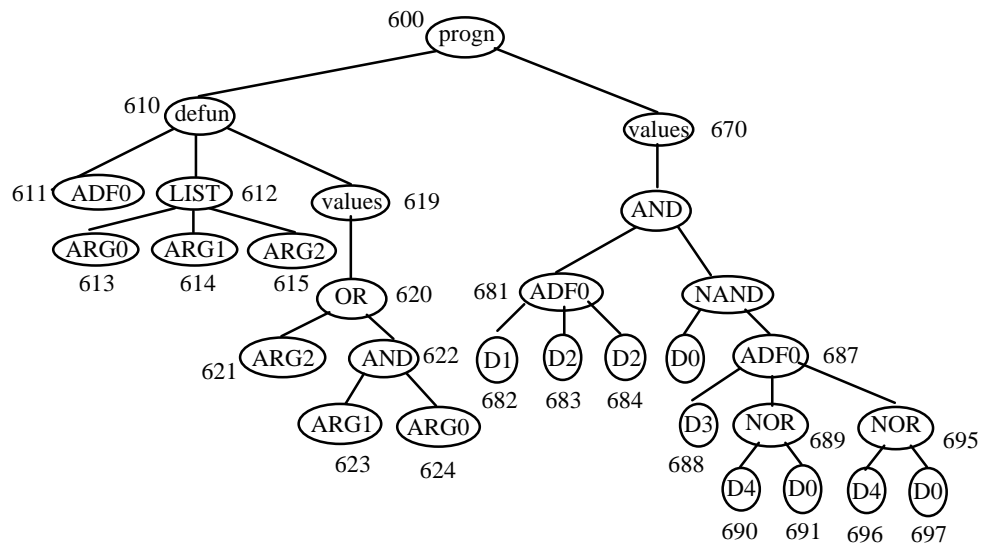
# PROGRAM WITH 1 TWO-ARGUMENT AUTOMATICALLY DEFINED FUNCTION (ADF0) AND 1 RESULT-PRODUCING BRANCH – ARGUMENT MAP OF {2}



# PROGRAM WITH ARGUMENT MAP OF $\{2, 2\}$ CREATED USING THE OPERATION OF BRANCH DUPLICATION



# PROGRAM WITH ARGUMENT MAP OF {3} CREATED USING THE OPERATION OF ARGUMENT DUPLICATION



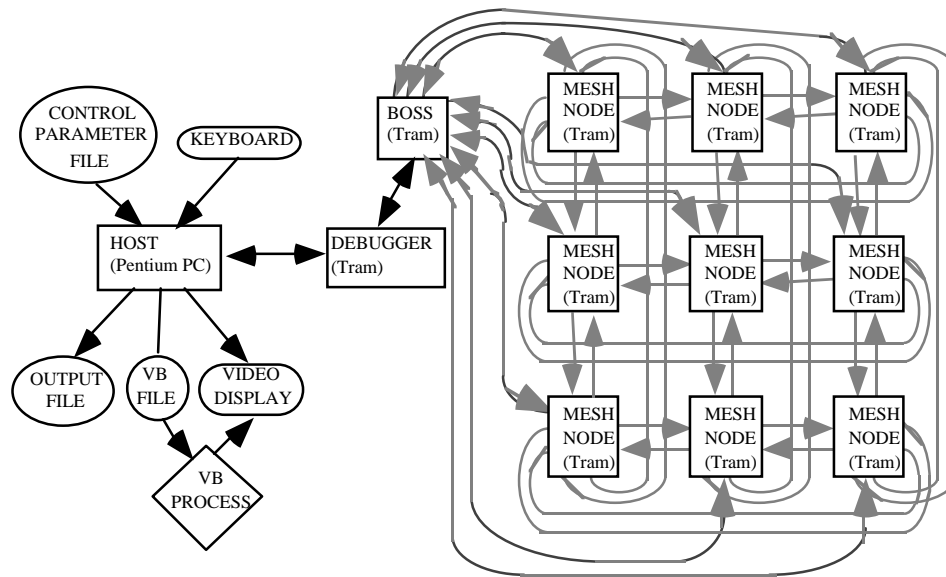
## **PARALLELIZATION OF GA OR GP**

- **The computational burden of GA or GP is concentrated in the task of measuring the fitness for each fitness case for each individual in the population for each generation**
  - Time-consuming primitive functions
  - Lengthy simulations
  - Complicated state transition calculations
  - Different initial conditions
  - Series of probabilistic experiments
- **However, not coupled**

# PARALLELIZATION OF GA OR GP

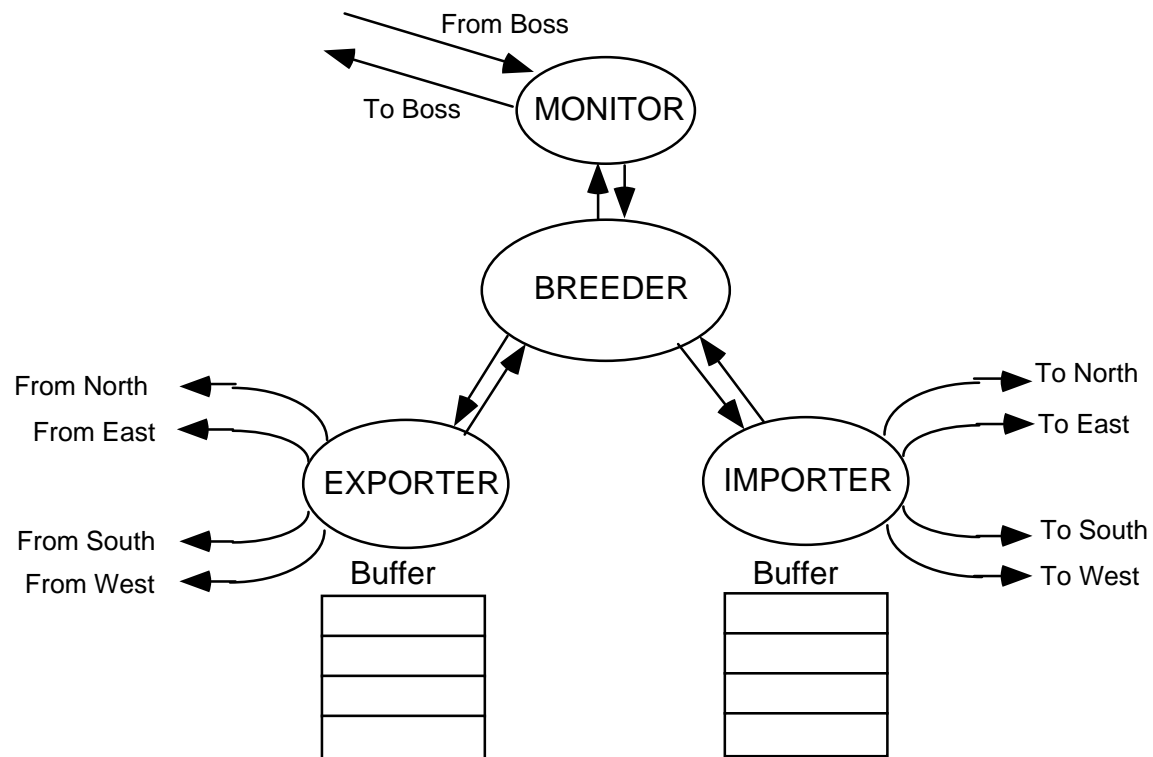
- **By individuals in the population**
  - Timing (Program size, Simulation time)
- **By fitness cases**
  - Timing (Simulation time, Protein length)
  - Matching between hardware and problem
- **By "Demes" ("Island" model)**
  - No synchronization of islands
  - Occasional small amounts of migration (low band width requirement for communication)
  - Emigrants go (fitness-based selection)
  - Immigrants arrive and are absorbed (fitness-based making of space)
  - Fault-tolerant

# PARALLEL GA / GP SYSTEM



- **3 lines of 2-way communication exist within the overall system**
  - between the Host and the Boss process,
  - between the Boss process and all 64 (only 9 shown above) processing nodes of the network, and
  - between (toroidally) adjacent processing nodes of the network.

# THE FOUR INTER-COMMUNICATING PROCESSES OF EACH OF THE 64 PROCESSING NODES



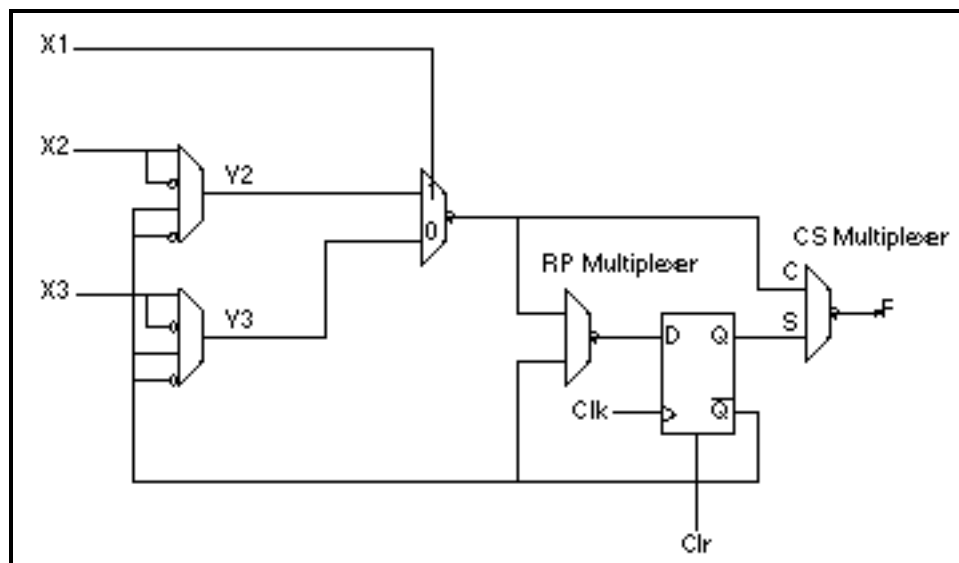
## **NOVEL WAYS TO MEASURE FITNESS**

- Evaluate fitness by operating a tethered robot in actual environment for a certain period of time (e.g., 30 seconds)**
- Evaluate fitness of electrical filter circuit by connecting computer to a bread-board with the circuit and collecting outputs for particular inputs**
- Evaluate fitness by using field-programmable gate arrays (FPGA) to process thousands (or millions) of fitness cases (combinations of inputs) at very high speed**
- Laboratory experiments lasting a day or longer for each fitness case of each generation of a run**

# EVOLVABLE HARDWARE

## RAPIDLY RECONFIGURABLE FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

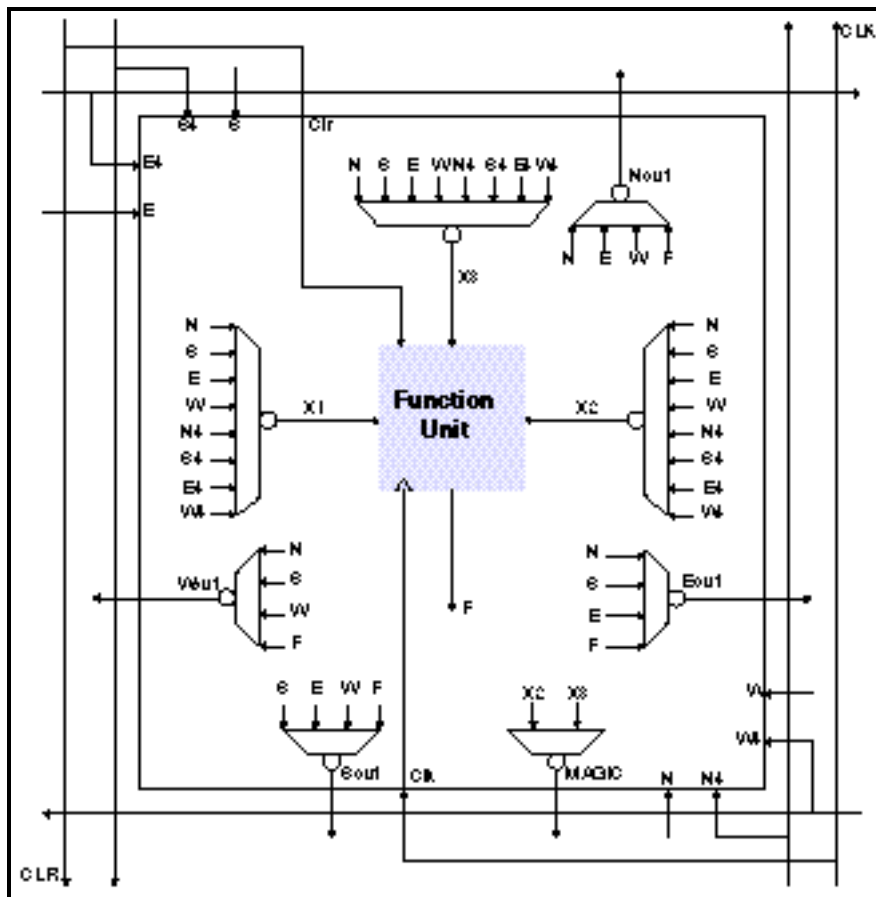
### FUNCTION UNIT FOR ONE CELL OF THE XILINX XC6216 CHIP



# EVOLVABLE HARDWARE

## RAPIDLY RECONFIGURABLE FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

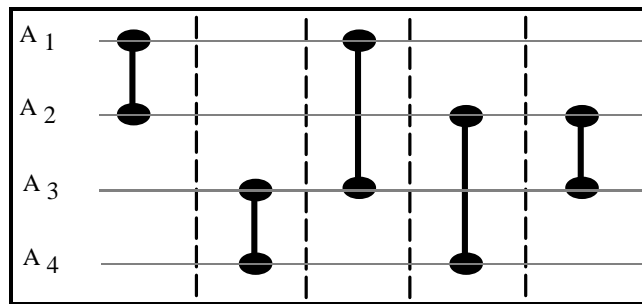
### ONE CELL OF THE XILINX XC6216 CHIP



# EVOLVABLE HARDWARE

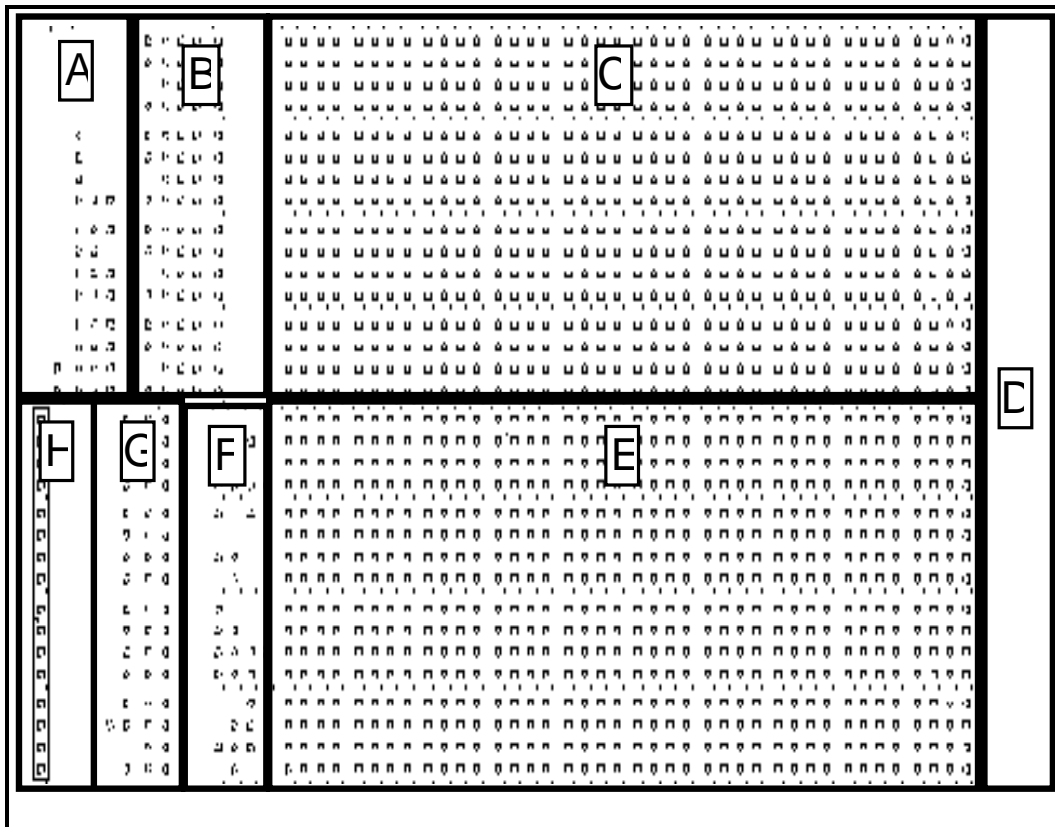
## RAPIDLY RECONFIGURABLE FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

### SORTING NETWORKS



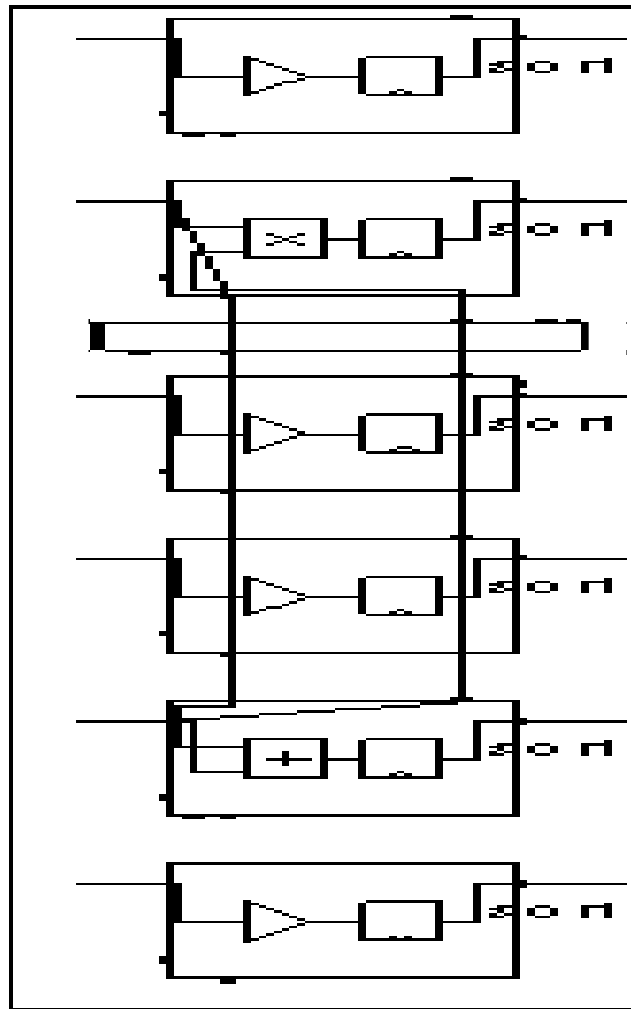
# EVOLVABLE HARDWARE

## 32 × 64 PORTION OF THE XILINIX XC6216 CHIP FOR SORTING NETWORKS



# EVOLVABLE HARDWARE

## IMPLEMENTATION OF (COMPARE- EXCHANGE 2 5) IN TOP 6 CELLS OF ONE $1 \times 16$ VERTICAL COLUMN OF THE XILINIX XC6216 CHIP FOR SORTING NETWORKS



# **EVOLVABLE HARDWARE**

## **RAPIDLY RECONFIGURABLE FIELD- PROGRAMMABLE GATE ARRAYS (FPGAs)**

### **SORTING NETWORKS**

- **A 16-step 7-sorter was evolved that has two fewer steps than the sorting network described in O'Connor and Nelsons' patent (1962) and that has the same number of steps as the 7-sorter that was devised by Floyd and Knuth subsequent to the patent and described in Knuth 1973.**
- **Evolved reversible sorting networks for 4, 5, and 6 items for Mathemania contest.**

# **IMPLEMENTATION OF GP IN ASSEMBLY CODE – COMPILED GENETIC PROGRAMMING SYSTEM (PETER NORDIN 1994)**

- **Opportunity to speed up GP that is done by slowly INTERPRETING GP program trees. Instead of interpreting the GP program tree, EXECUTE this sequence of assembly code.**
- **Can identify small set of primitive functions that is useful for large group of problems, such as +, -, \*, % and also use some conditional operations (IFLTE), some logical functions (AND, OR, XOR, XNOR) and perhaps others (e.g., SRL, SLL, SETHI from Sun 4).**
- **Then, generate random sequence of assembly code instructions at generation 0 from this small set of machine code instructions (referring to certain registers).**
- **If ADFs are involved, generate fixed header and footer of function and appropriate function call.**
- **Perform crossover possibly so as to preserve the integrity of subtrees.**
- **If ADFs**

**are involved, perform crossover so as to preserve the integrity of the header and footer of function and the function call.**

# DISCOVERY BY GENETIC PROGRAMMING OF A CELLULAR AUTOMATA RULE THAT IS BETTER THAN ANY KNOWN RULE FOR THE MAJORITY CLASSIFICATION PROBLEM

Rule	Accuracy	Test cases
Best rule evolved by bit-string genetic algorithm (Das, Mitchell, and Crutchfield 1994)	76.9%	$10^6$
GKL 1978 human-written	81.6%	$10^6$
Davis 1995 human-written	81.8%	$10^6$
Das 1995 human-written	82.178%	$10^7$
Best rule evolved by genetic programming (in this paper)	82.326%	$10^7$

Andre, David, Bennett III, Forrest H., and Koza, John R. 1996. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors).

***Genetic Programming 1996: Proceedings of the First Annual Conference,  
July 28-31, 1996, Stanford University. Cambridge, MA: The MIT Press.***

# THE SUCCESSION OF "BEST" CELLULAR AUTOMATA RULES FOR THE MAJORITY CLASSIFICATION TASK, PRESENTED IN TRUTH TABLE ORDER FROM 00000000 TO 11111111 (I.E. 0 TO 127)

Rule	State Transitions
<b>GKL 1978</b> human-written	00000000 01011111 00000000 01011111 00000000 01011111 00000000 01011111 00000000 01011111 11111111 01011111 00000000 01011111 11111111 01011111
<b>Davis 1995</b> human-written	00000000 00101111 00000011 01011111 00000000 00011111 11001111 00011111 00000000 00101111 11111100 01011111 00000000 00011111 11111111 00011111
<b>Das (1995)</b> human-written	00000111 00000000 00000111 11111111 00001111 00000000 00001111 11111111 00001111 00000000 00000111 11111111 00001111 00110001 00001111 11111111
<b>Best rule</b> evolved by genetic programm ing (in this paper)	00000101 00000000 01010101 00000101 00000101 00000000 01010101 00000101 01010101 11111111 01010101 11111111 01010101 11111111 01010101 11111111

# **TRANSMEMBRANE SEGMENT IDENTIFICATION PROBLEM**

- **Goal is to classify a given protein segment (i.e., a subsequence of amino acid residues from a protein sequence) as being a transmembrane domain or non-transmembrane area of the protein (without using biochemical knowledge concerning hydrophobicity typically used by human-written algorithms for this task).**
- **Four different versions of genetic programming have been applied to this problem. The performance of all four versions using genetic programming is slightly superior to that of algorithms written by knowledgeable human investigators.**

**Koza, John R. and Andre, David. 1996b. Evolution of iteration in genetic programming. In Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming. Cambridge, MA: MIT Press. In Press.**

# **AUTOMATED DISCOVERY OF MOTIFS IN PROTEIN DATABASES**

- **Genetic programming successfully evolved motifs for detecting the D-E-A-D box family of proteins and for detecting the manganese superoxide dismutase family. Both motifs were evolved without prespecifying their length. Both evolved motifs employed automatically defined functions to capture the repeated use of common subexpressions. The two genetically evolved consensus motifs detect the two families either as well as, or slightly better than, the comparable human-written motifs found in the PROSITE database.**

**Koza, John R. and Andre, David. 1996c. Automatic discovery of protein motifs using genetic programming. In Yao, Xin (editor). 1996. *Evolutionary Computation: Theory and Applications*. Singapore: World Scientific. In Press.**

## COMPUTER USAGE

- **Power PC 601 processor =  $8 \times 10^7$  Hertz**
- **times 64 =  $5.12 \times 10^9$  ops/second**
- **times 86,400 sec. =  $4.4 \times 10^{14}$  ops / day**
- **time 2 days =  $10^{15}$  ops**

## HUMAN BRAIN

- **$10^{12}$  neurons operating at  $10^3$  per second =  $10^{15}$  ops per second**
- **1 bs =  $10^{15}$  ops**

## PETAFLIPS COMPUTING INITIATIVE

- **$10^{15}$  operations PER SECOND by 2010**

# **AUTOMATED "WYWIWYG" DESIGN OF ELECTRICAL CIRCUITS USING GENETIC PROGRAMMING**

- **Filters**
  - Lowpass, highpass, bandpass LC filters
  - Asymmetric passband Nielsen filter
  - Crossover (woofer-tweeter) filter
  - Crossover (woofer-midrange-tweeter)
  - Matching Butterworth, Chebychev, elliptic
- **Amplifiers (5 dB, 60 dB, 96 dB)**
- **Computational Circuits**
  - Squaring circuit
  - Cubing circuit
  - Square root circuit
  - Cube root circuit
- **Source Identification Problem**
  - Three-way
  - Four-way (with changing environment)
- **Circuit for time-optimal fly-to controller**
- **Temperature-sensing circuit**
- **Voltage reference circuit**

# **DIFFICULTY OF AUTOMATED CIRCUIT DESIGN**

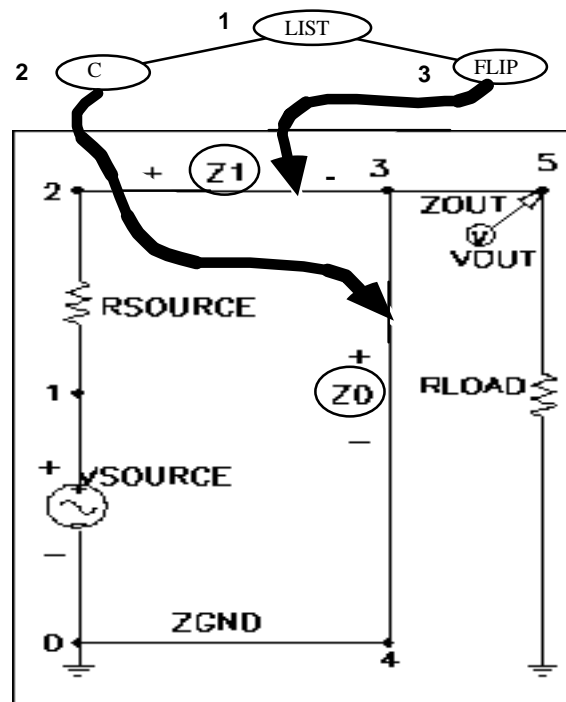
- **A very hard problem**
  - Exponential in the number of components
  - More than  $10^{300}$  circuits with a mere 20 components
- **An important problem**
  - Too few analog designers
  - There is an "Egg-shell" of analog circuitry around almost all digital circuits
  - Analog circuits must be redesigned with each new generation of process technology
- **No existing automated techniques**
  - In contrast with digital
  - Existing analog techniques do only sizing of components, but do not create the topology
- **In analog design, the search is for a satisfactory circuit, but not necessarily an optimal one**

# **USE OF AUTOMATICALLY DEFINED FUNCTIONS AND ARCHITECTURE- ALTERING OPERATIONS IN THE DESIGN OF ELECTRICAL CIRCUITS**

- **Automatically defined functions and architecture-altering operations**
  - Two-Band Crossover (Woofer and Tweeter) Filter (with ADFs and architecture-altering operations)
  - Lowpass filter (with ADFs)
- **Lowpass filter (with ADFs and architecture-altering operations)**
  - Double-Bandpass Filter (with ADFs and architecture-altering operations) – "Comb" filters

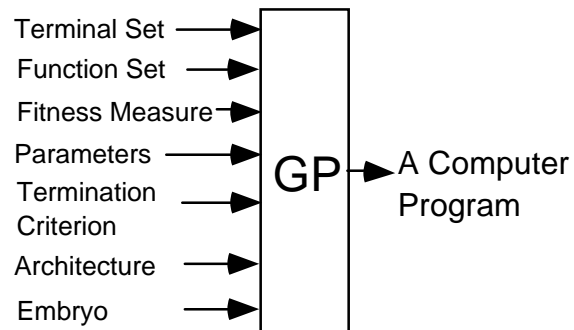
# EMBRYO WITH TWO MODIFIABLE WIRES Z0 AND Z1 AND CIRCUIT-CONSTRUCTING PROGRAM TREE (WITH TWO RESULT-PRODUCING BRANCHES) WITH TWO WRITING HEADS POINTING TO THE TWO MODIFIABLE WIRES

- Embryonic electrical circuit
- Circuit-constructing program trees
  - Component-creating functions
  - Connection-creating functions



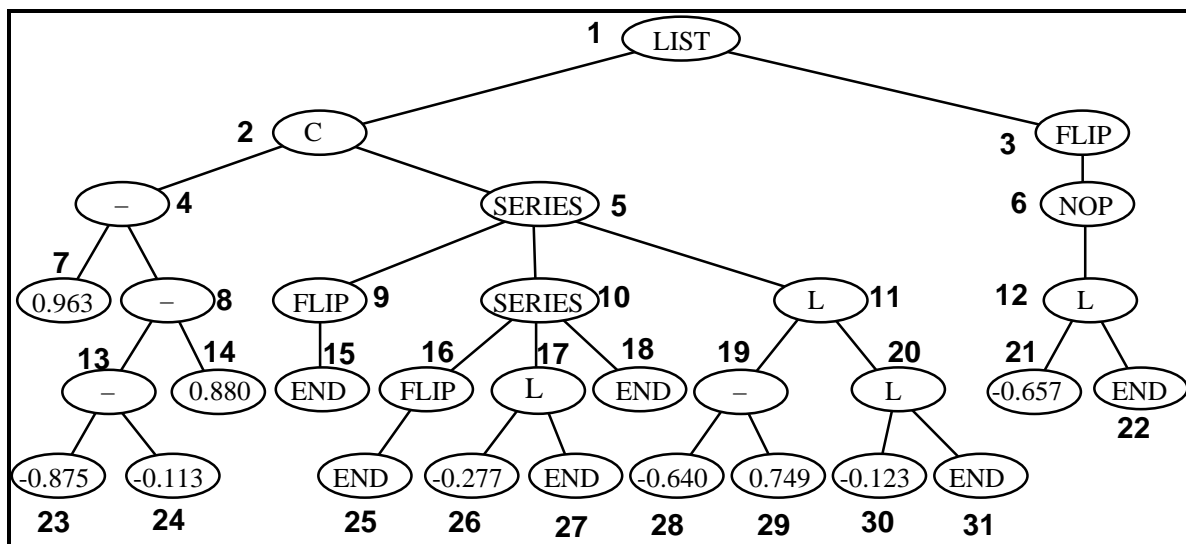
# DEVELOPMENT OF A CIRCUIT FROM A CIRCUIT-CONSTRUCTING PROGRAM TREE AND THE EMBRYONIC CIRCUIT

- **Circuit-constructing program trees**
  - Component-creating functions
  - Connection-creating functions
- **Embryonic electrical circuit**

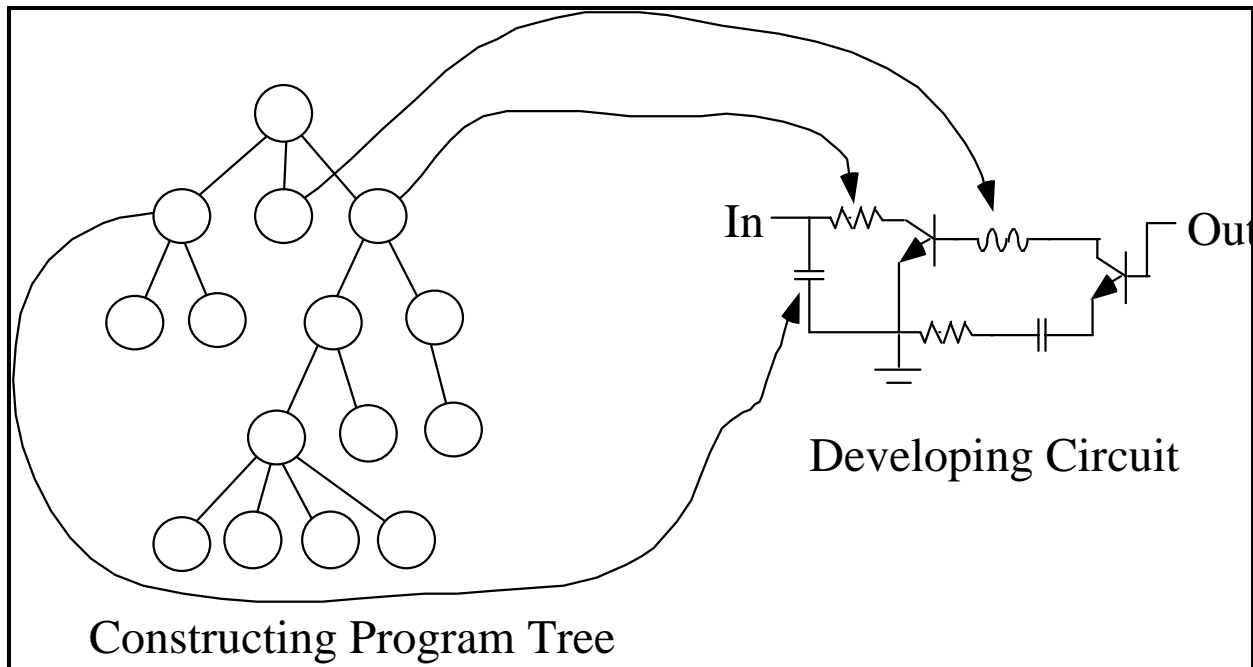


# EXAMPLE OF CIRCUIT- CONSTRUCTING PROGRAM TREE

```
(LIST (C (- 0.963 (- (- -0.875
-0.113) 0.880)) (series (flip
end) (series (flip end) (L -
0.277 end) end) (L (- -0.640
0.749) (L -0.123 end)))) (flip
(nop (L -0.657 end)))))
```

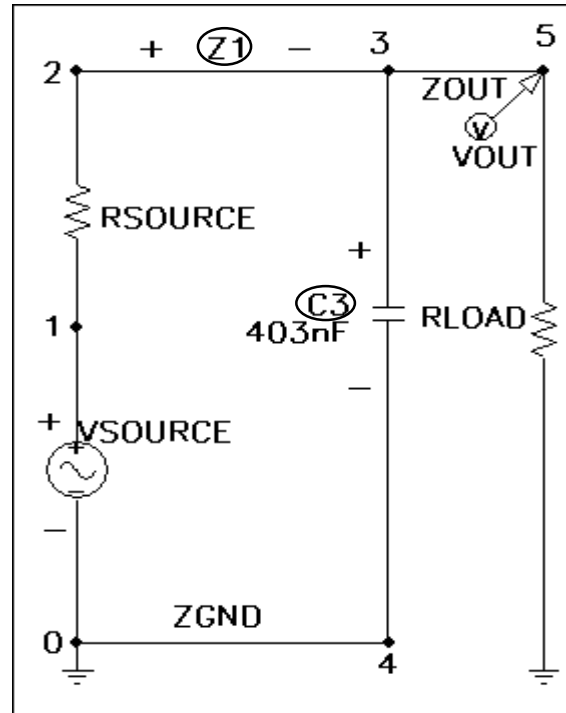


# DEVELOPMENT OF A CIRCUIT



- Each function in the circuit-constructing tree acts on a part of the circuit and changes it in some way (e.g. creates a capacitor, creates a parallel structure, adds a connection to ground, etc)
- A “writing head” points from each function to the part of the circuit that the function will act on.
- Each function inherits writing heads from its parent in the tree

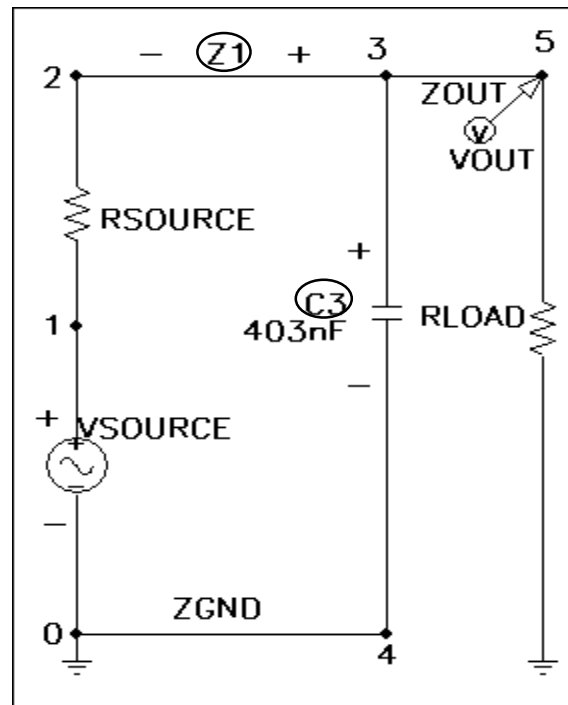
## RESULT OF THE c (2) FUNCTION



```
(LIST (C (- 0.963 (- (- -0.875
-0.113) 0.880)) (series (flip
end) (series (flip end) (L -
0.277 end) end) (L (- -0.640
0.749) (L -0.123 end)))) (flip
(nop (L -0.657 end))))
```

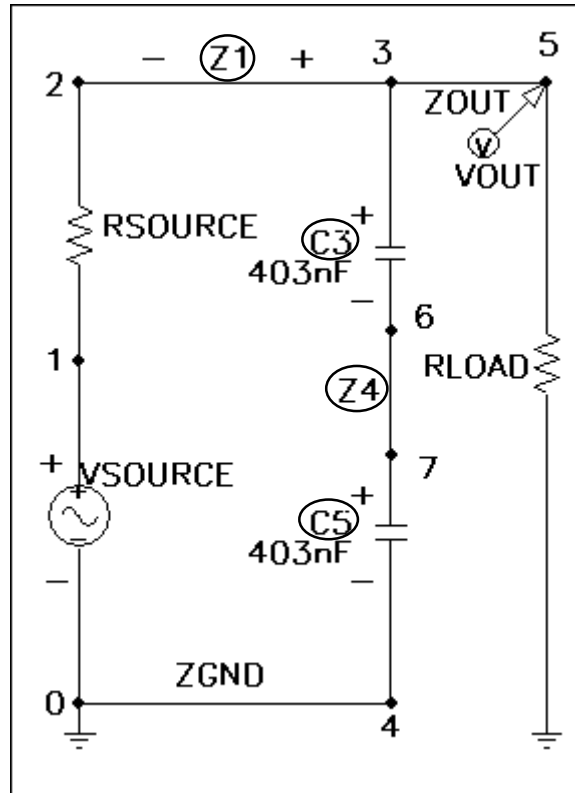
**NOTE:** Interpretation of arithmetic value

## RESULT OF THE FLIP (3) – IN 2nd RESULT-PRODUCING BRANCH



```
(LIST (C (- 0.963 (- (- -0.875
-0.113) 0.880)) (series (flip
end) (series (flip end) (L -
0.277 end) end) (L (- -0.640
0.749) (L -0.123 end)))) (flip
(nop (L -0.657 end))))
```

## RESULT OF SERIES (5) FUNCTION

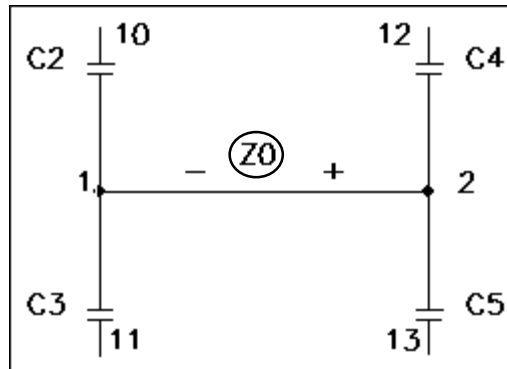


```
(LIST (C (- 0.963 (- (- -0.875
-0.113) 0.880)) (series (flip
end) (series (flip end) (L -
0.277 end) end) (L (- -0.640
0.749) (L -0.123 end)))) (flip
(nop (L -0.657 end))))
```

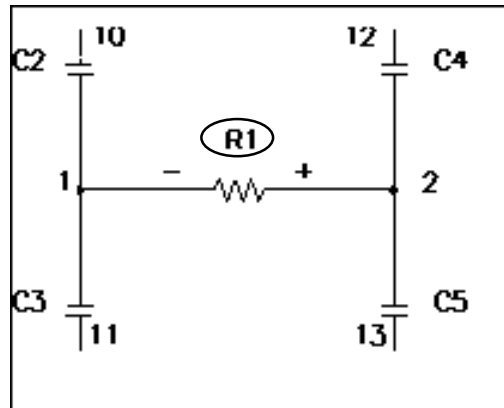
## **COMPONENT-CREATING FUNCTIONS**

- **Resistor R function**
- **Capacitor C function**
- **Inductor L function**
- **Diode D function**
- **Transistor QT0 function**
- **Digital AND0 function**
- **Transformer TRANSFORMER0 function**

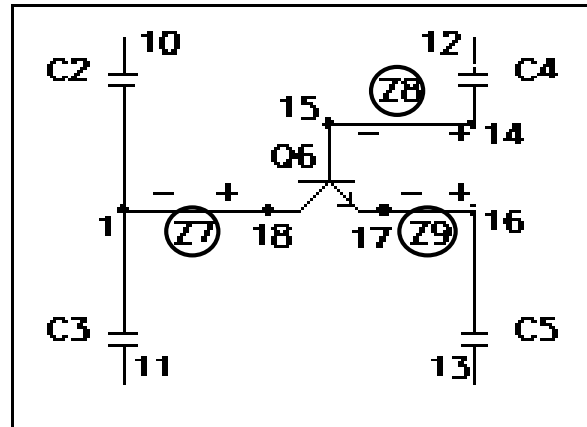
## A CIRCUIT CONTAINING A MODIFIABLE WIRE Z0



## RESULT AFTER R FUNCTION



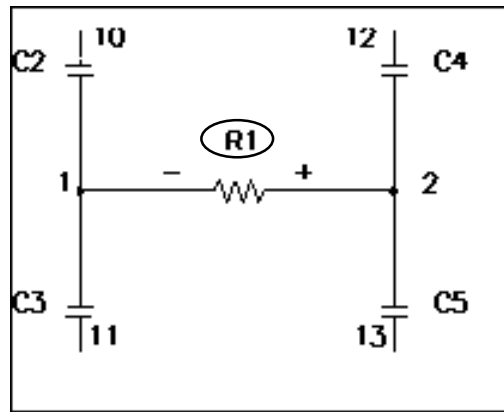
## RESULT AFTER Q<sub>T0</sub> FUNCTION



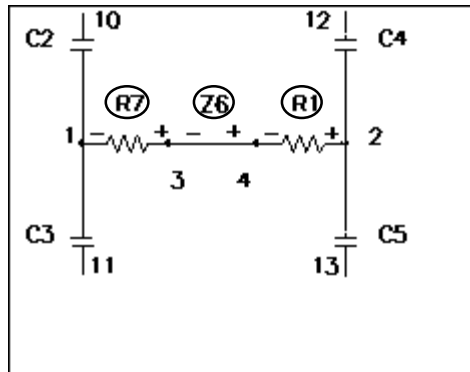
## **CONNECTION-CREATING FUNCTIONS**

- **SERIES division function**
- **PSS and PSL parallel division functions**
- **STAR1 division function**
- **TRIANGLE1 division function**
- **VIA0 function**
- **FLIP function**

# A CIRCUIT CONTAINING A RESISTOR R1



## AFTER THE SERIES FUNCTION



## **NETLIST WITH R1**

R1 2 1 5ohms

**C2 1 10**

**C3 1 11**

**C4 2 12**

**C5 2 13**

## **NETLIST AFTER SERIES FUNCTION**

R1 2 4 5ohms

Z6 4 3

R7 3 1 5ohms

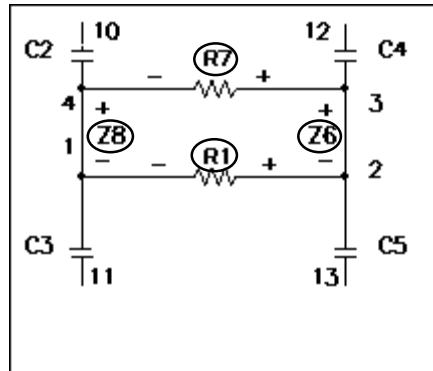
**C2 1 10**

**C3 1 11**

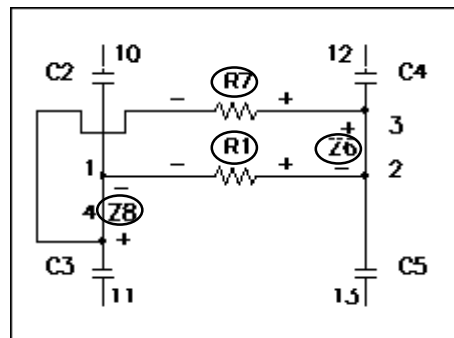
**C4 2 12**

**C5 2 13**

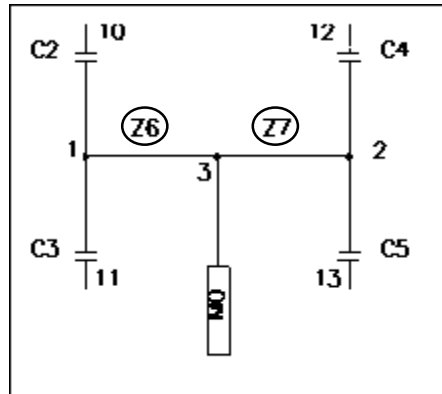
## AFTER PSS PARALLEL DIVISION



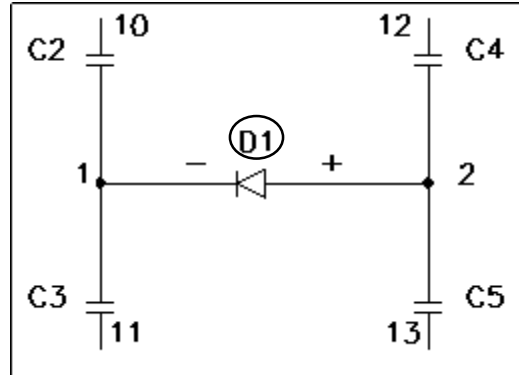
## AFTER PSL PARALLEL DIVISION



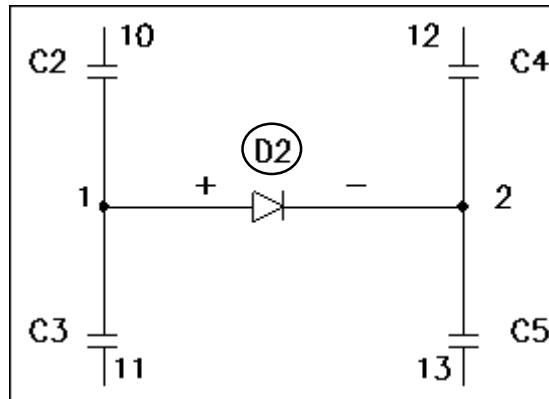
## AFTER VIA0 FUNCTION



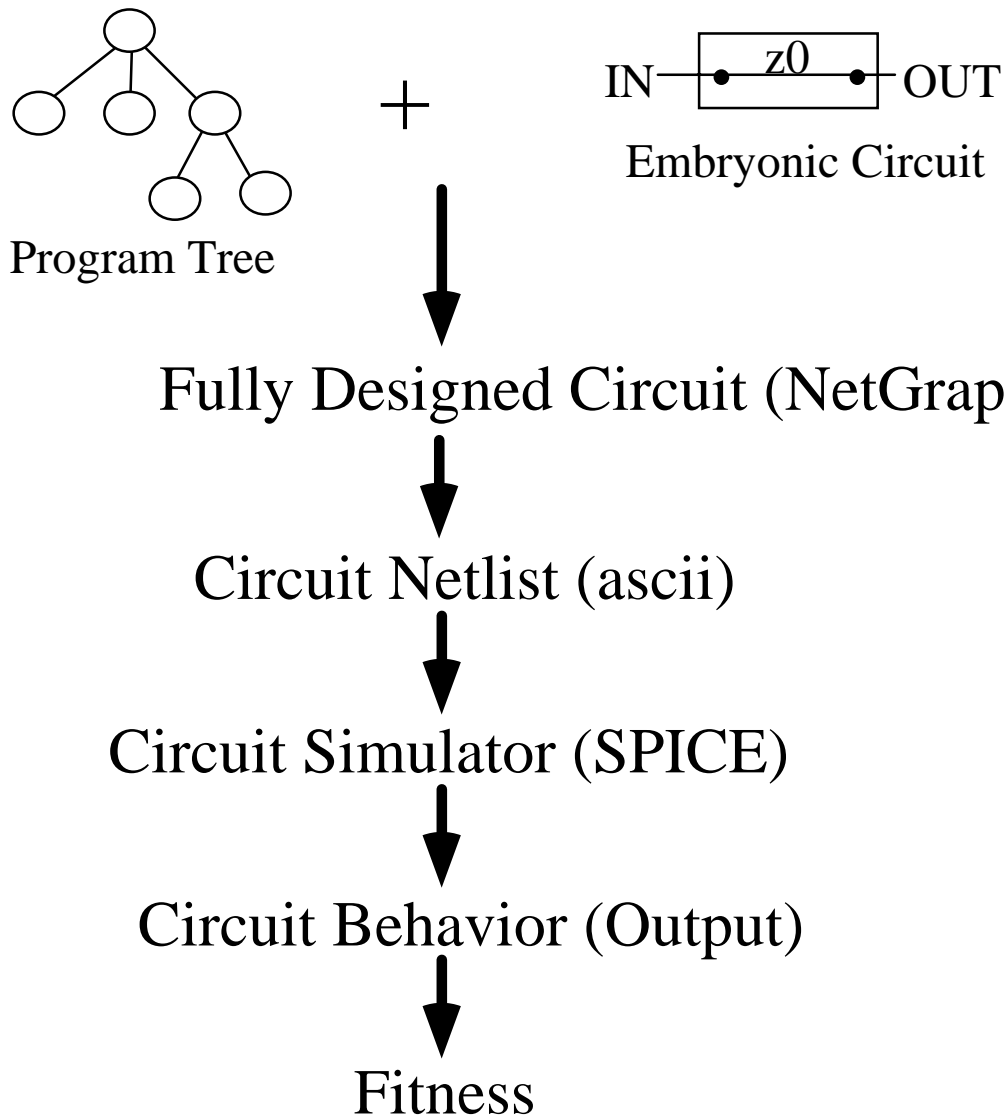
## A CIRCUIT CONTAINING A DIODE D1



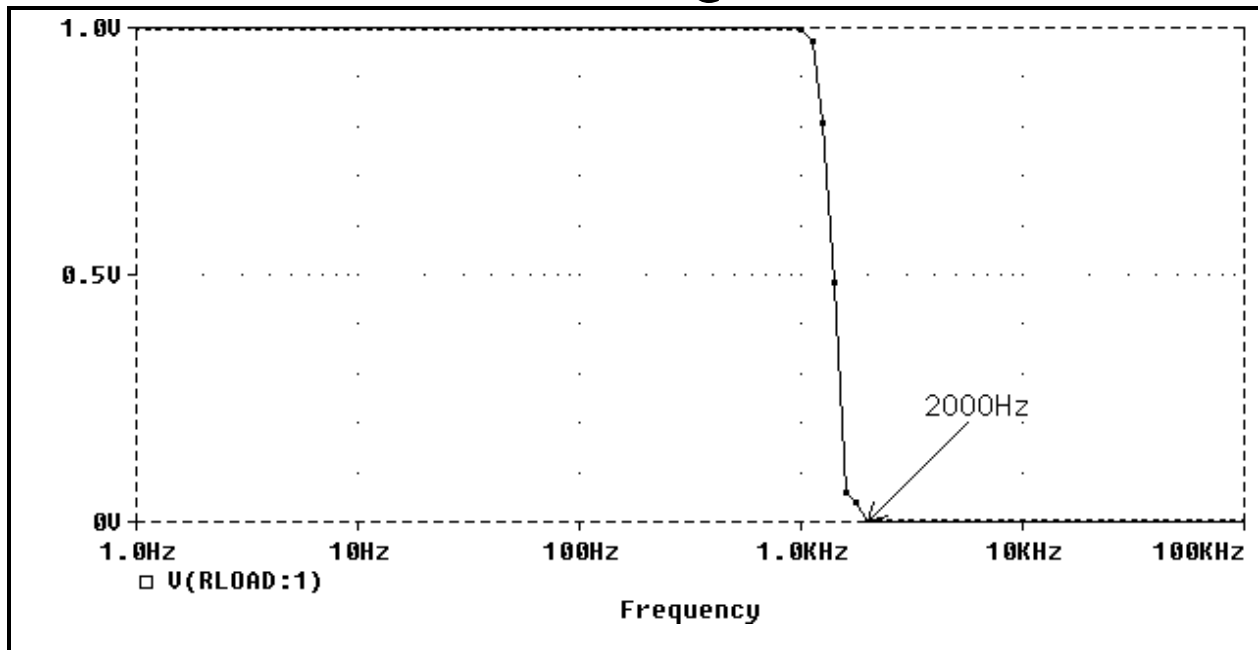
## AFTER THE FLIP FUNCTION



## EVALUATION OF THE FITNESS OF A CIRCUIT



# BEHAVIOR OF A LOWPASS FILTER VIEWED IN THE FREQUENCY DOMAIN



- Examine circuit's behavior for each of 101 frequency values chosen over five decades of frequency (from 1 Hz to 100,000 Hz) with each decade divided into 20 parts (using a logarithmic scale). The fitness measure
  - does not penalize ideal values
  - slightly penalizes acceptable deviations
  - heavily penalizes unacceptable deviations
- Fitness is sum  $F(t) = \sum_{i=0}^{100} [W(f_i)]d(f_i)$ 
  - $f(i)$  is the frequency of fitness case  $i$
  - $d(x)$  is the difference between the target and observed values at frequency of fitness case  $i$

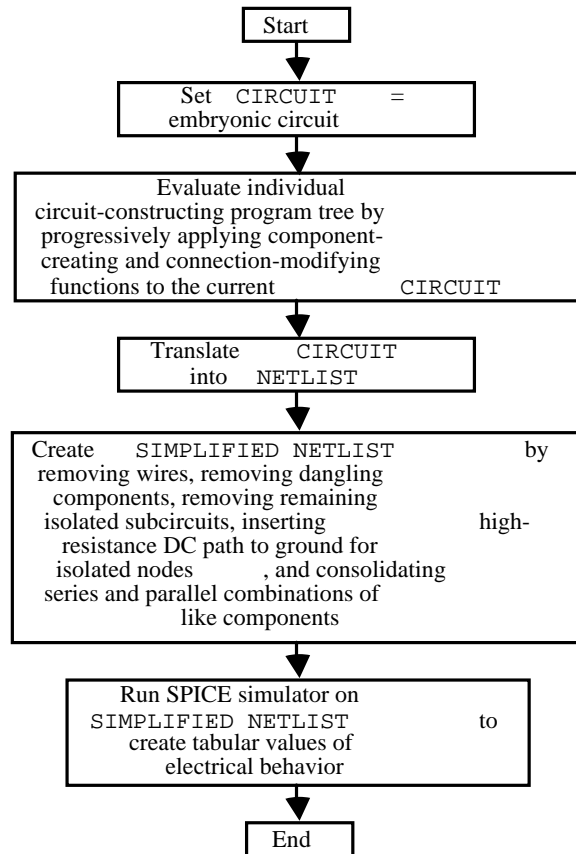
- $W(y,x)$  is the weighting at frequency  $x$

## **LOWPASS FILTER – CONTINUED**

- **61 points in the 3-decade interval from 1 Hz to 1,000 Hz**
  - For voltage equaling the ideal value of 1.0 volts, the deviation is **0.0**
  - For voltage between 970 and 1,000 millivolts, the absolute value of the deviation from 1,000 millivolts is weighted by a factor of **1.0**
  - For voltage less than 970 millivolts, the absolute value of the deviation from 1,000 millivolts is weighted by a factor of **10.0**
- **35 points from 2,000 Hz to 100,000 Hz**
  - For voltage equaling the ideal value of 0.0 volts, the deviation is **0.0**
  - For voltage between 0 millivolts and 1 millivolt, the absolute value of the deviation from 0 millivolts is weighted by a factor of **1.0**
  - For voltage above 1 millivolt, the absolute value of the deviation from 0 millivolts is weighted by factor of **10.0**

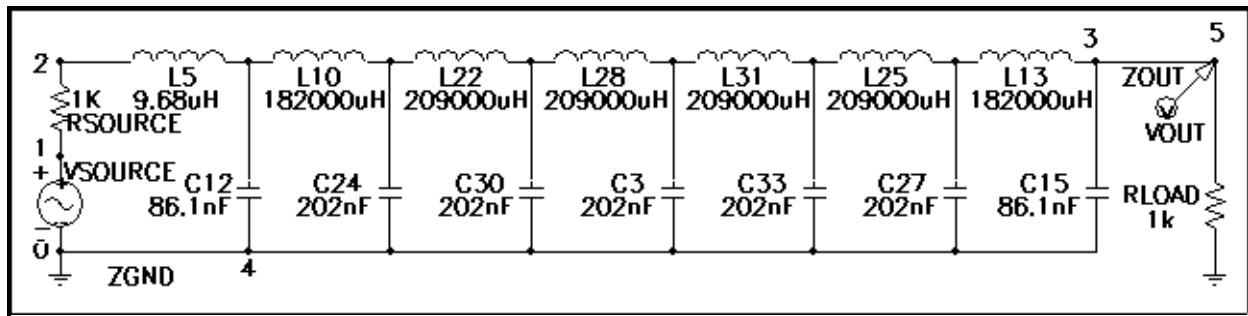
- **5 "don't care" points between 1,000 Hz and 2,000 Hz**
- **Unsimulatable programs =  $10^8$  penalty**
- **Hits is number (0–101) of compliant points (i.e., those getting weight of 1.0)**

# EVALUATION OF THE FITNESS OF A CIRCUIT

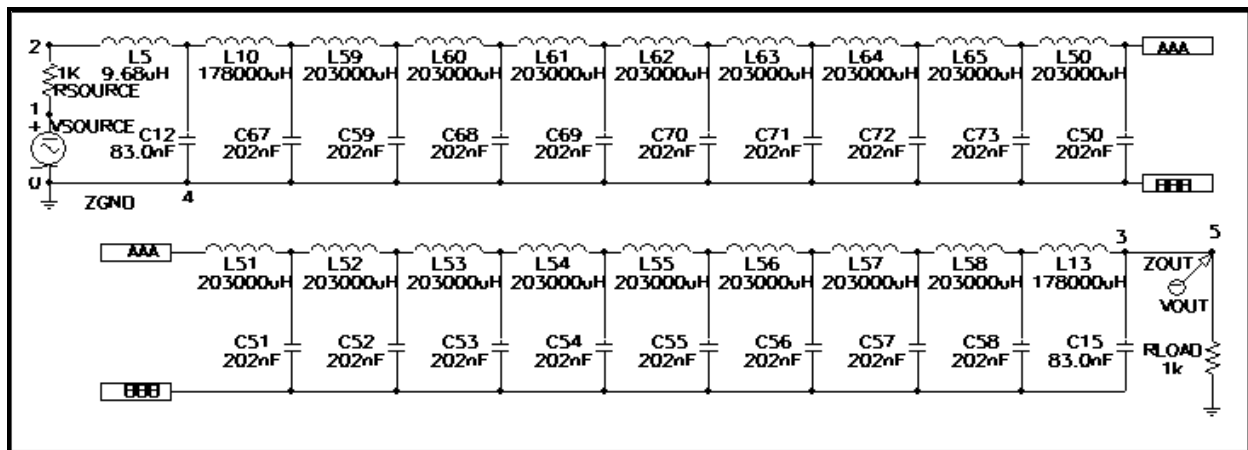


# DESIGN OF A LOWPASS FILTER

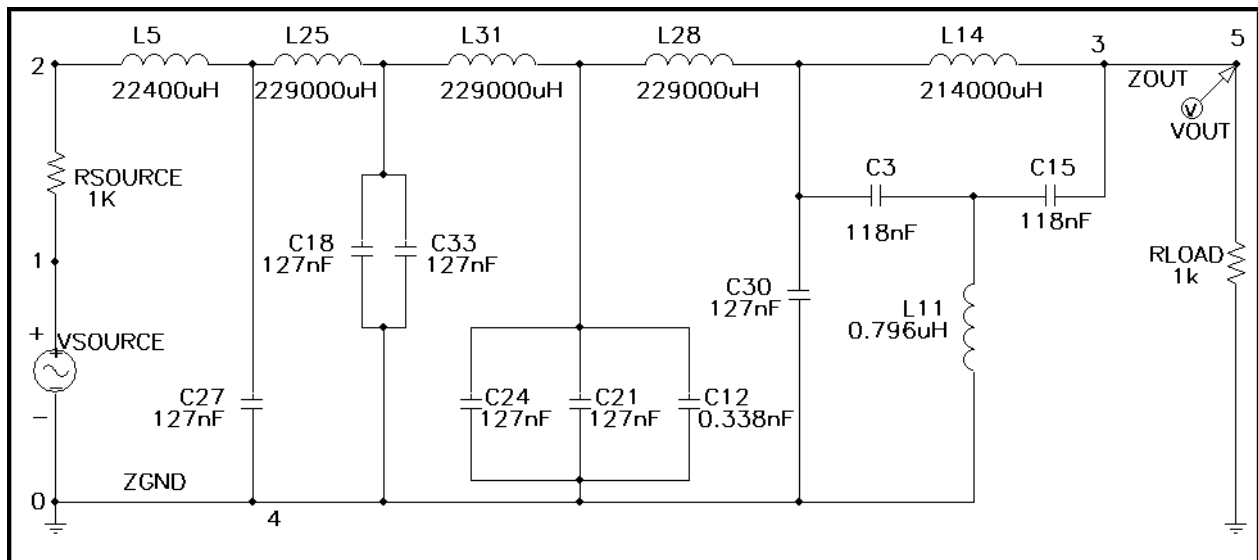
## 100%-COMPLIANT "7-RUNG LADDER"



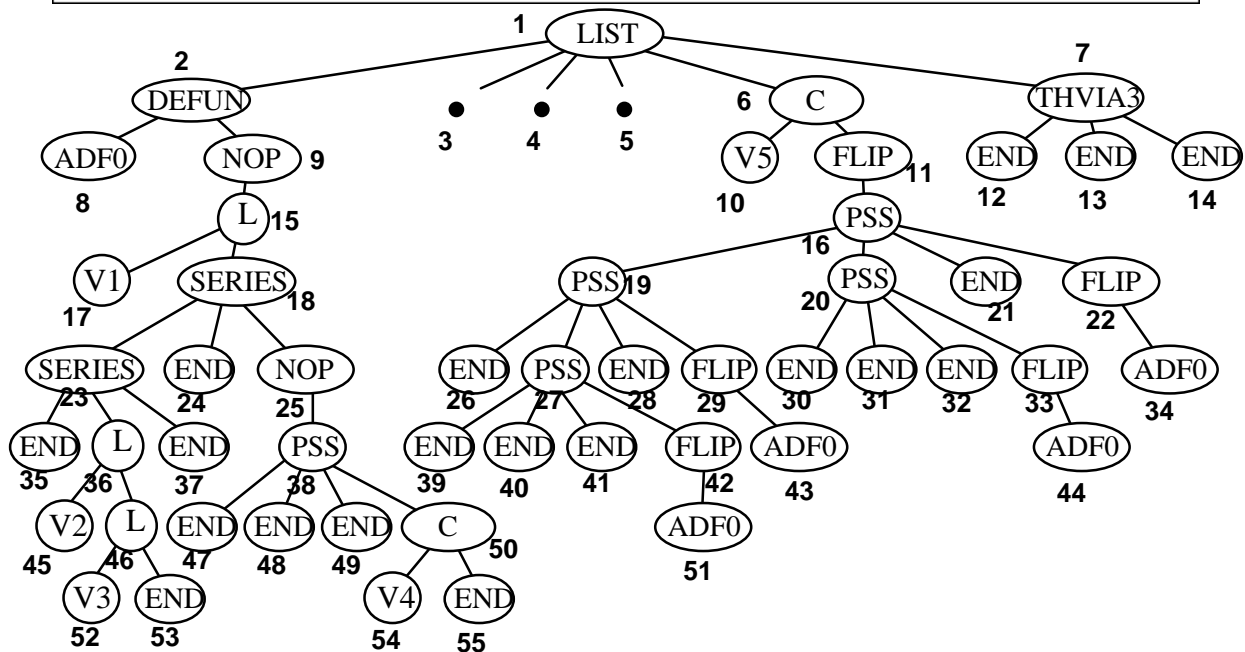
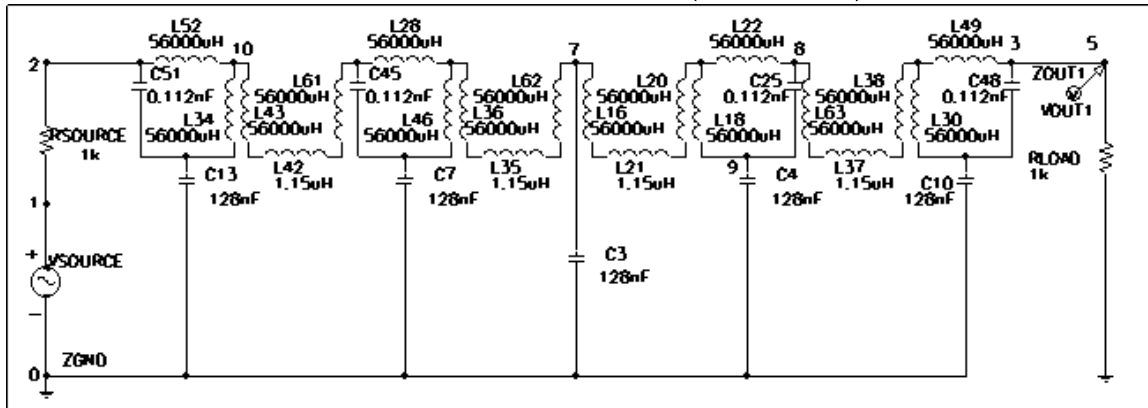
## 100%-COMPLIANT "19-RUNG LADDER"



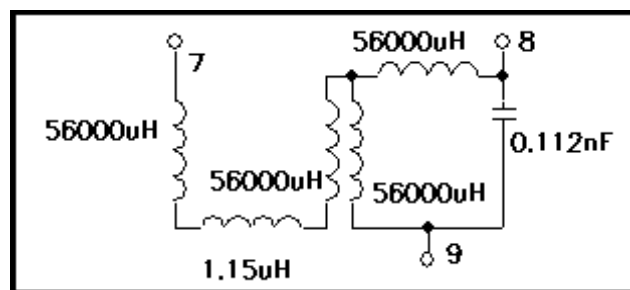
## 100%-COMPLIANT "BRIDGED T"



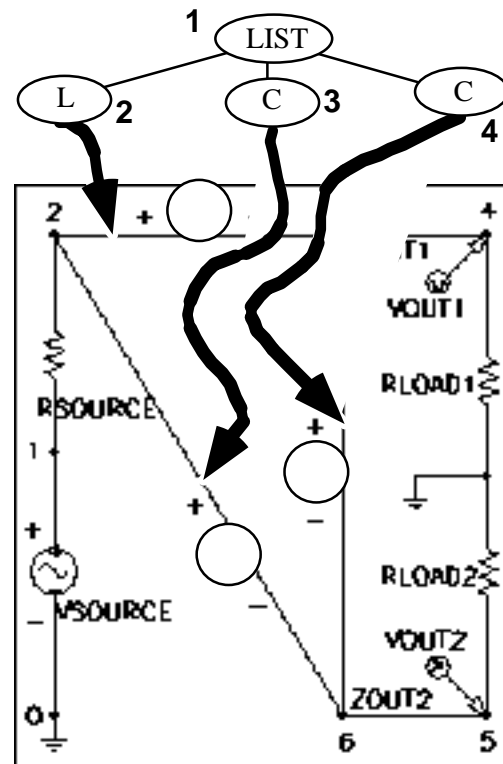
# DESIGN OF A LOWPASS FILTER USING AUTOMATICALLY DEFINED FUNCTIONS (ADFs)



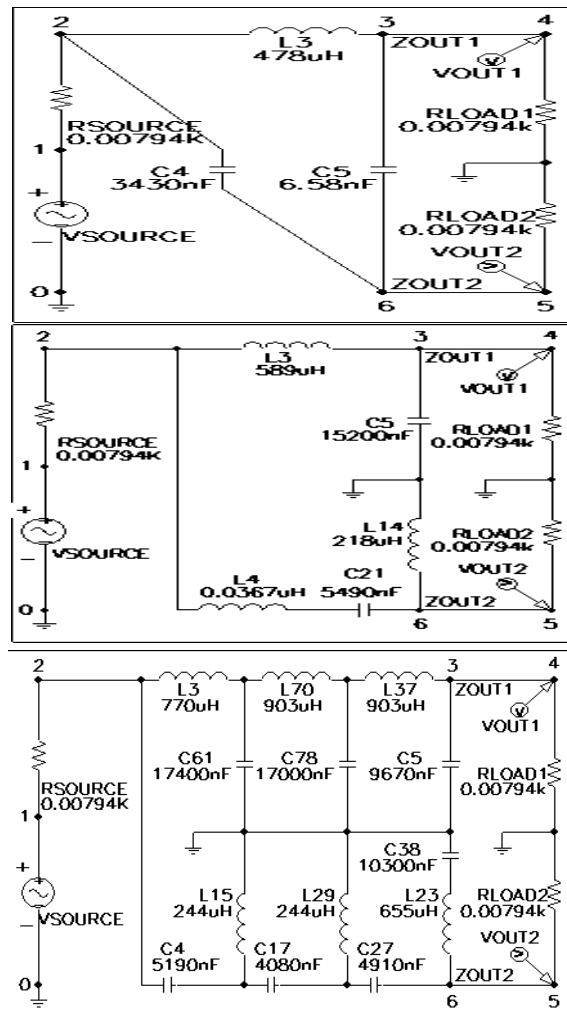
## EDITED VERSION OF ADF0



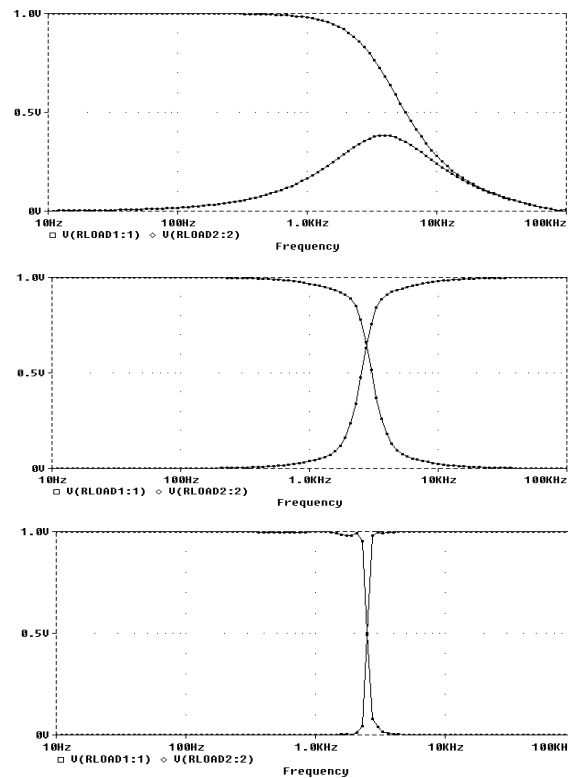
# EMBRYONIC CIRCUIT FOR A TWO-BAND CROSSOVER (WOOFER AND TWEETER) FILTER



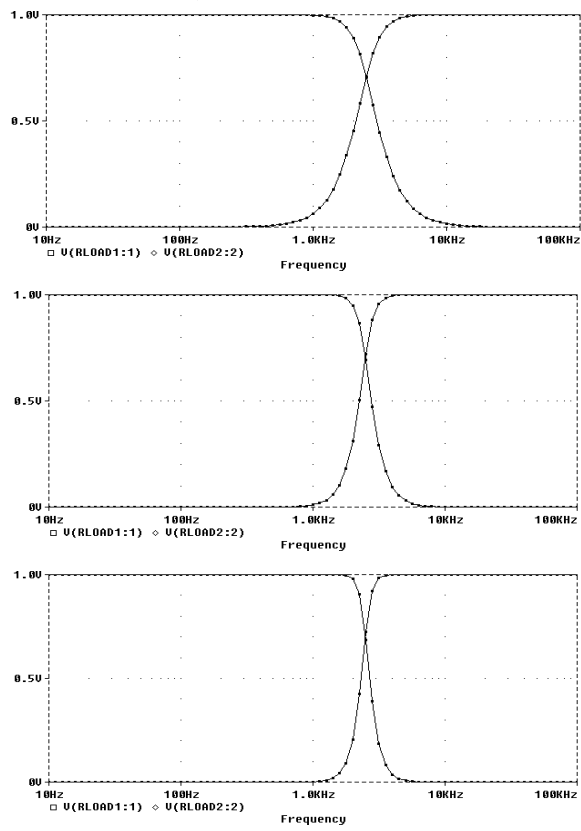
# DESIGN OF A TWO-BAND CROSSOVER (WOOFER AND TWEETER) FILTER BEST CIRCUIT OF GENERATIONS 0, 20, AND 137



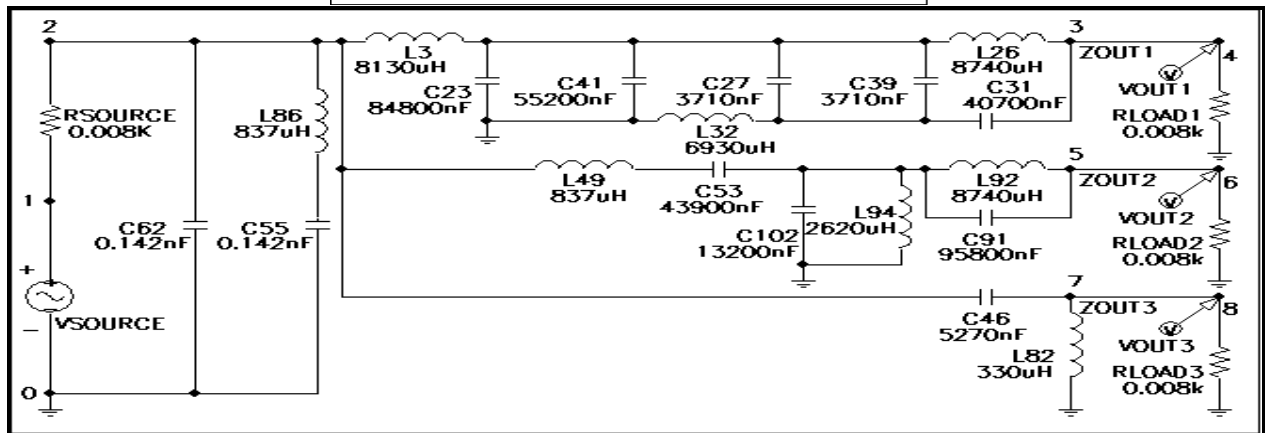
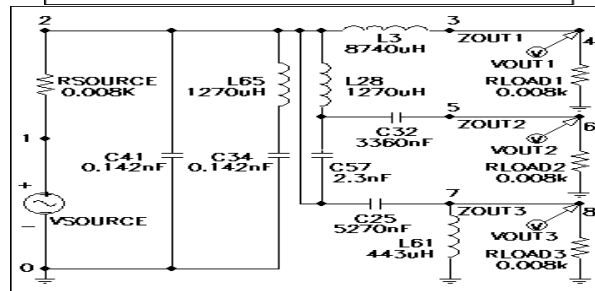
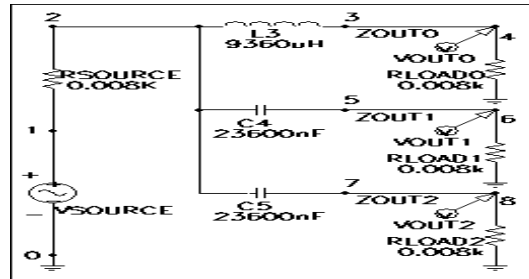
# DESIGN OF A TWO-BAND CROSSOVER (WOOFER AND TWEETER) FILTER FREQUENCY DOMAIN BEHAVIOR OF THE BEST CIRCUIT OF GENERATIONS 0, 20, AND 137



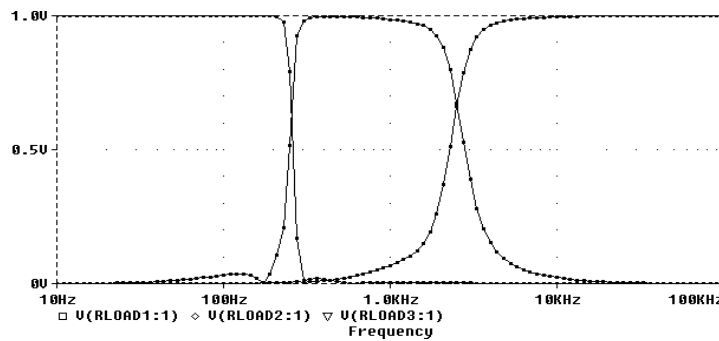
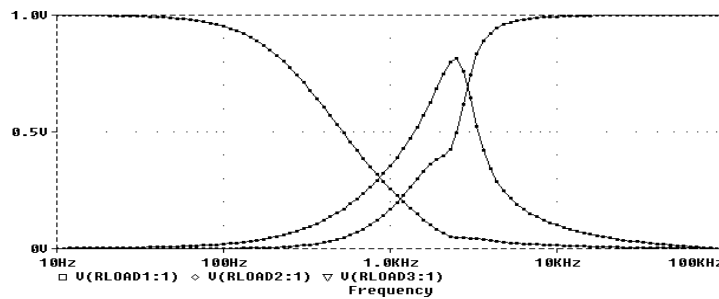
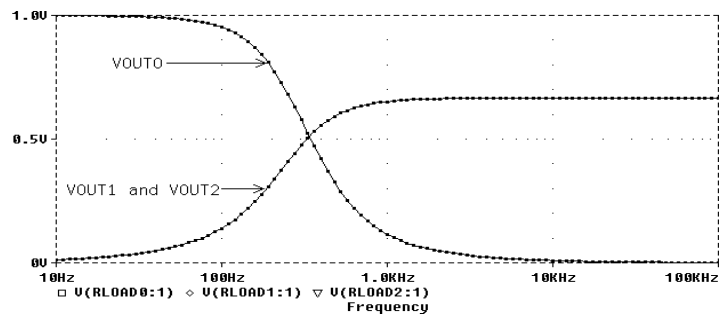
# FREQUENCY DOMAIN BEHAVIOR OF BUTTERWORTH FILTERS OF ORDER 3, 5, AND 7



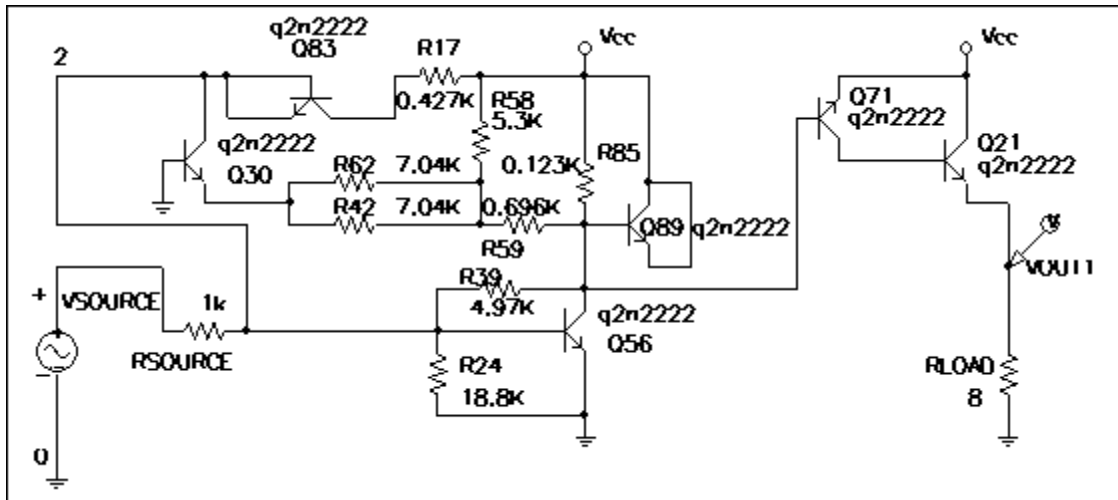
# DESIGN OF A THREE-BAND CROSSOVER (WOOFER-MIDRANGE- TWEETER) FILTER BEST CIRCUITS OF GENERATIONS 0, 54, AND 174



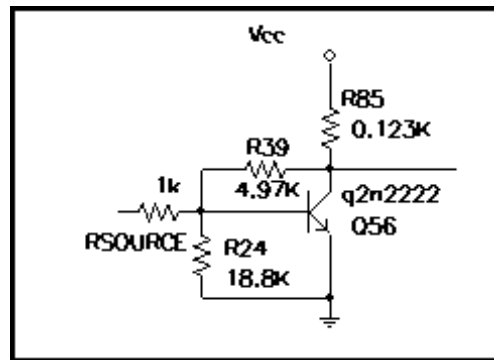
# DESIGN OF A THREE-BAND CROSSOVER (WOOFER-MIDRANGE- TWEETER) FILTER FREQUENCY DOMAIN BEHAVIOR OF THE BEST CIRCUIT OF GENERATIONS 0, 54, AND 174



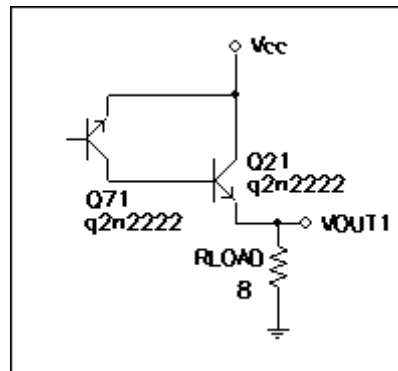
# GENETICALLY EVOLVED 5 DB AMPLIFIER FROM GENERATION 45



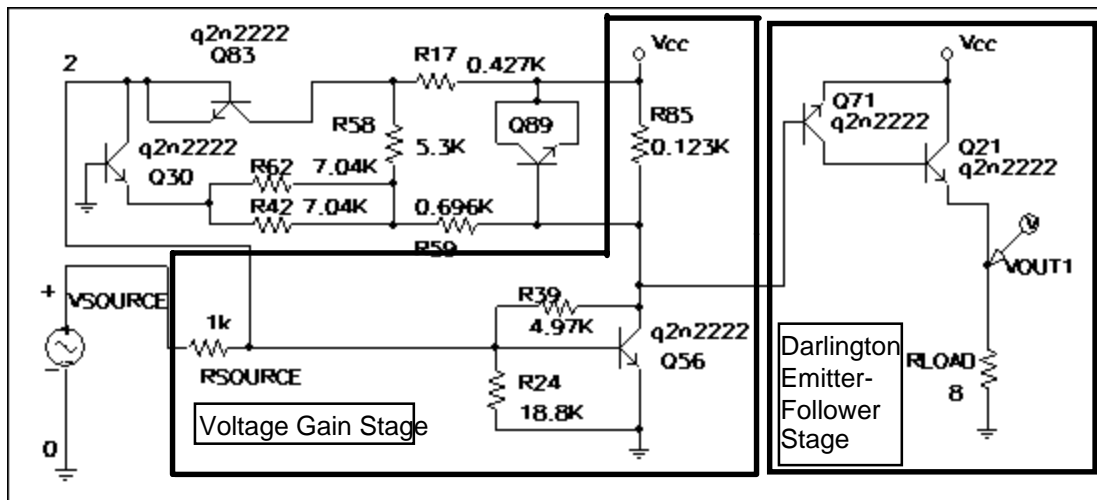
## VOLTAGE GAIN STAGE



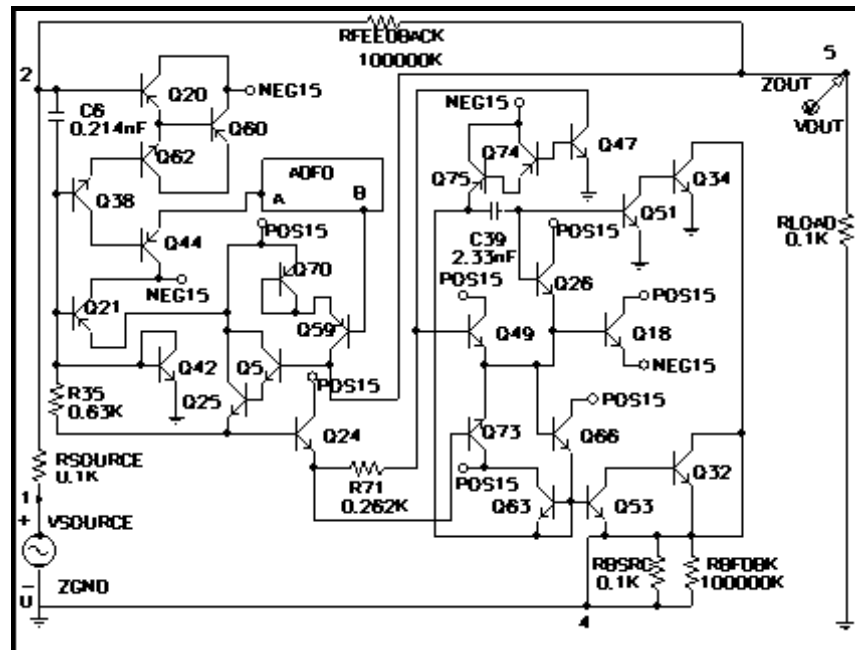
## DARLINGTON EMITTER FOLLOWER SECTION



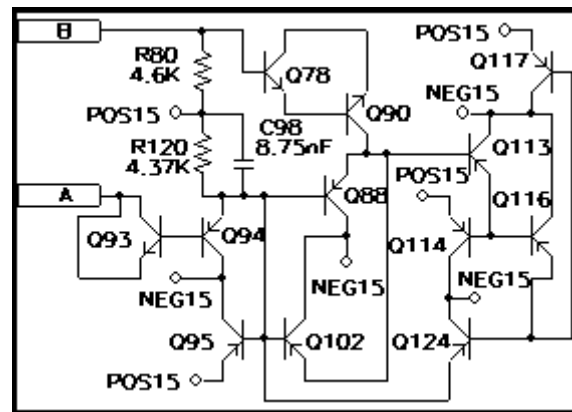
# **REDRAWN BEST-OF-GENERATION GENETICALLY EVOLVED 5 DB AMPLIFIER FROM GENERATION 45 SHOWING THE VOLTAGE GAIN STAGE AND DARLINGTON EMITTER FOLLOWER SECTION**



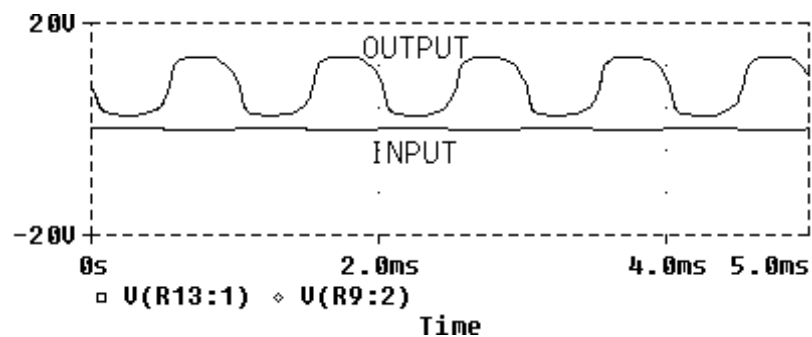
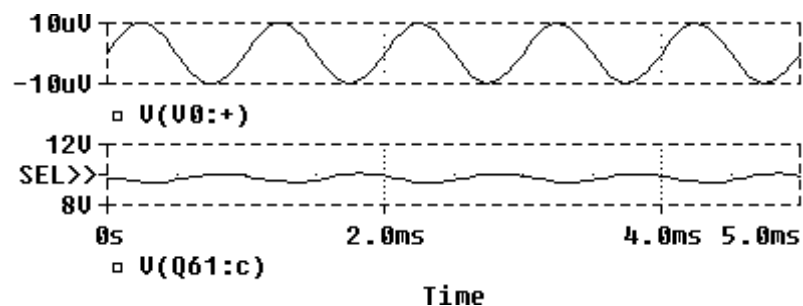
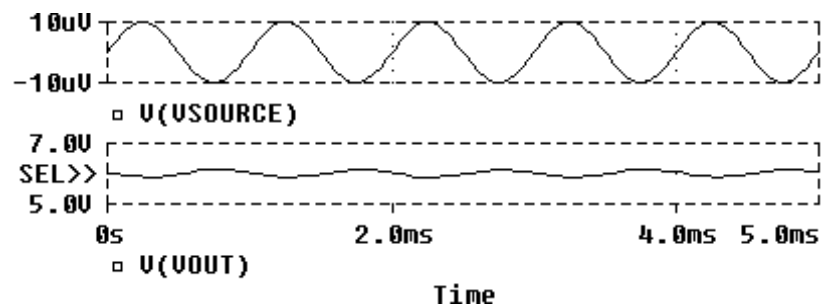
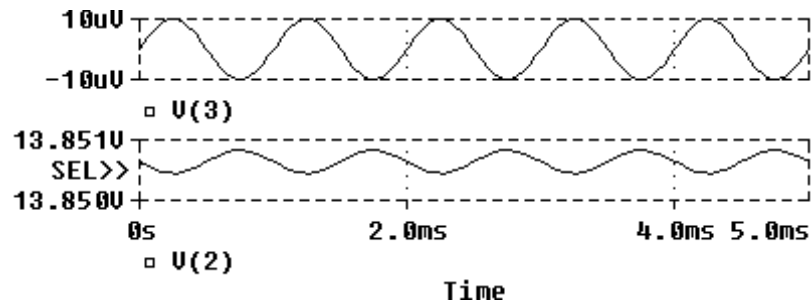
# 96 DB AMPLIFIER – BEST CIRCUIT OF GENERATION 86



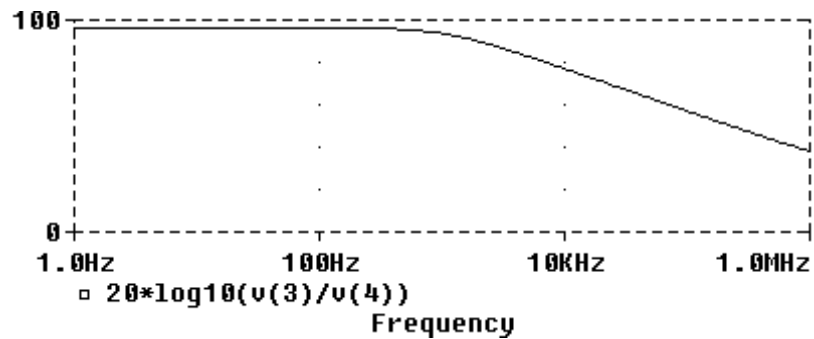
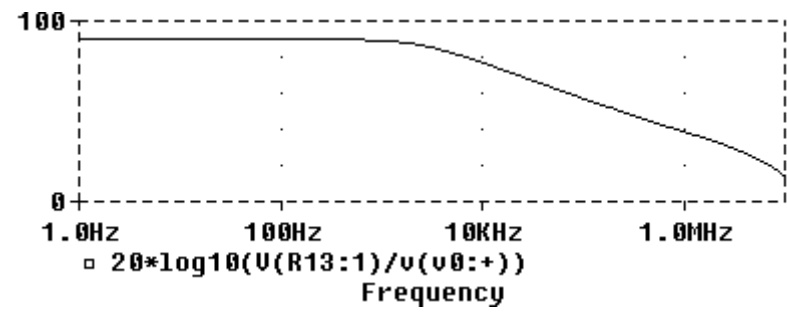
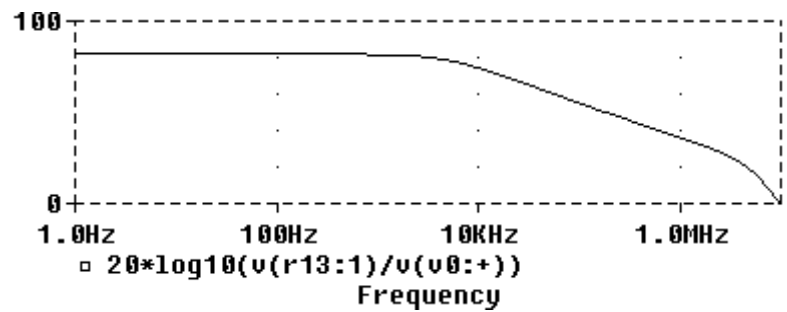
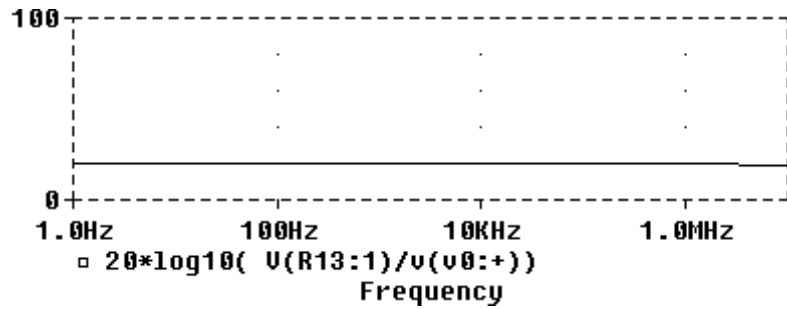
ADF0



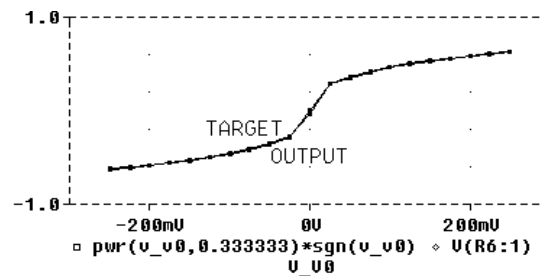
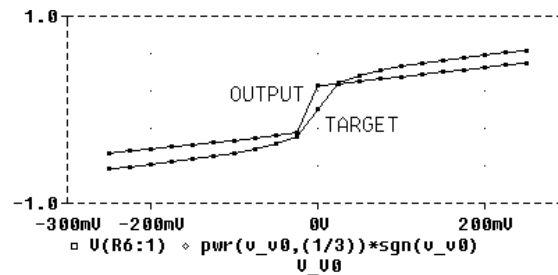
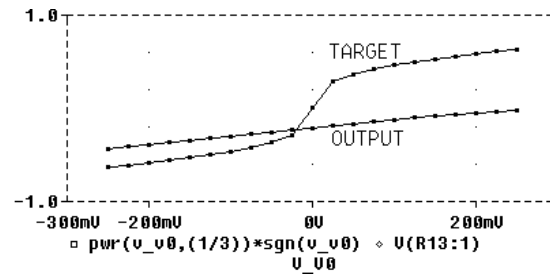
# 96 DB AMPLIFIER – BEST OF GENS 0, 42, 50, 86



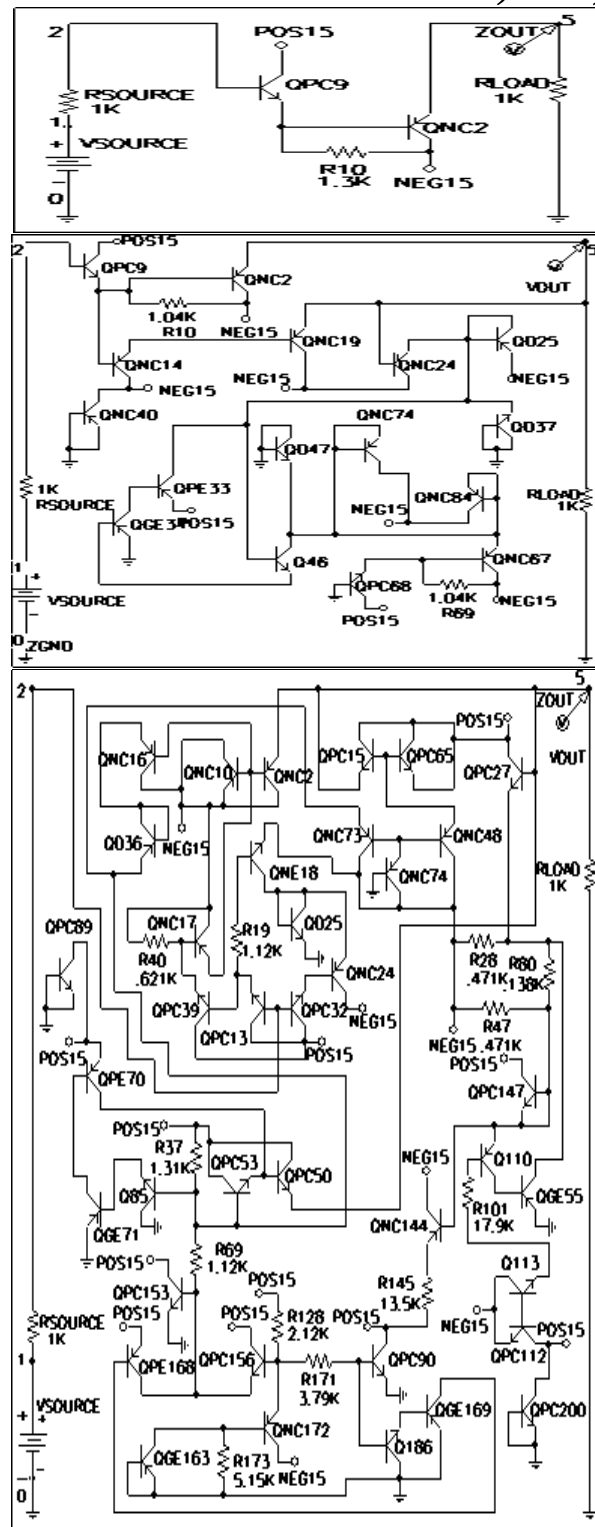
# 96 DB AMPLIFIER – AC SWEEPS – BEST OF GENS 0, 42, 50, 86



# CUBE ROOT COMPUTATIONAL CIRCUIT FROM GENERATIONS 0, 17, 60



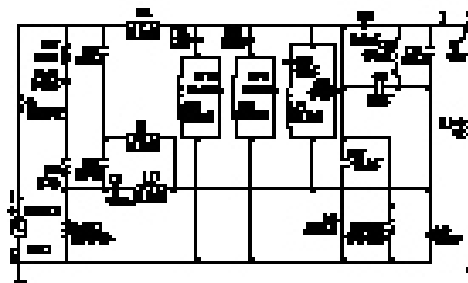
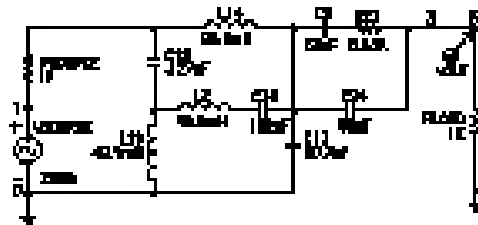
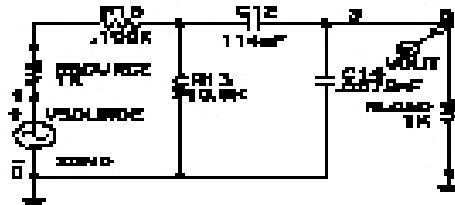
# CUBE ROOT COMPUTATIONAL CIRCUIT – GENS 0, 17, 60



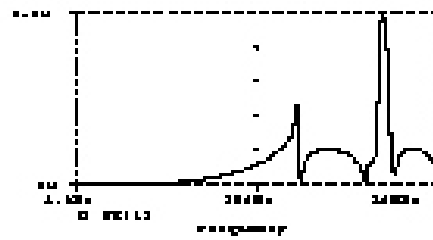
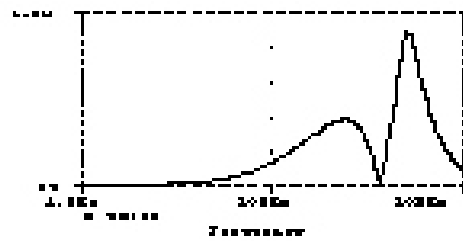
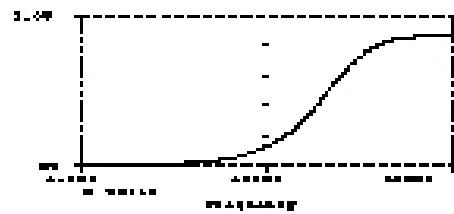
# OTHER COMPUTATIONAL CIRCUITS

- **Square root circuit**
- **Squaring circuit**
- **Cubing circuit**

## THREE-WAY SOURCE IDENTIFICATION PROBLEM – GENERATIONS 0, 20, 106

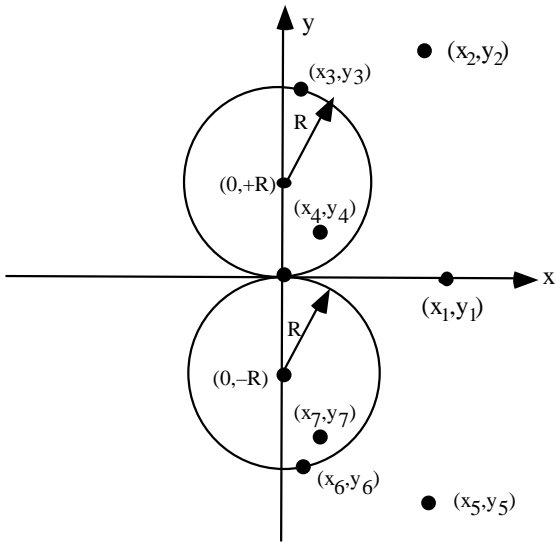


# THREE-WAY SOURCE IDENTIFICATION PROBLEM – GENERATIONS 0, 20, 106

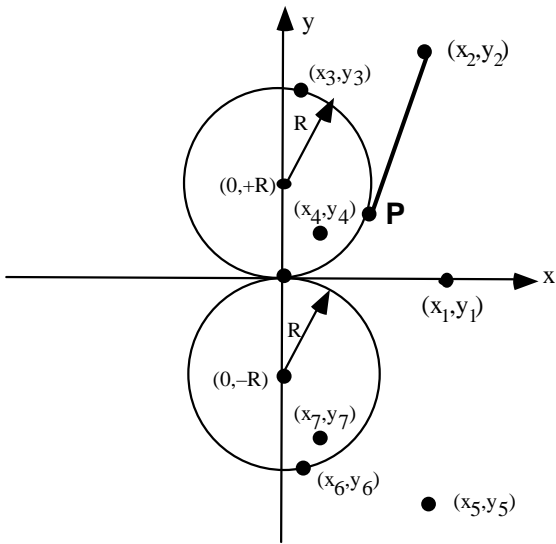


# TIME-OPTIMAL FLY-TO PROBLEM

## CASES 1 AND 2

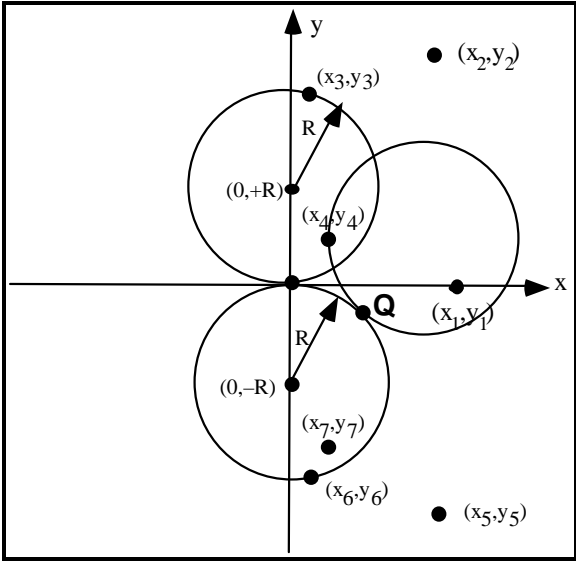


## CASE 3

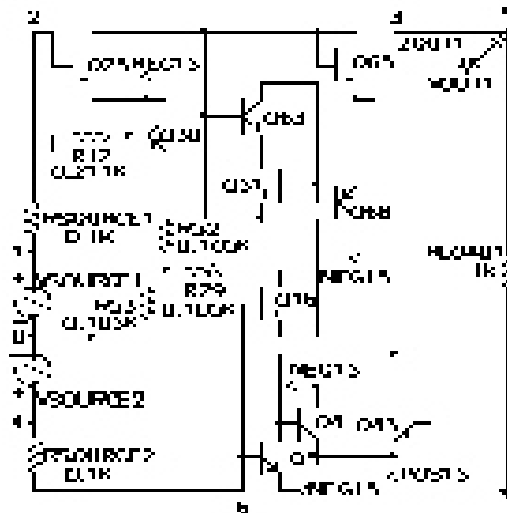
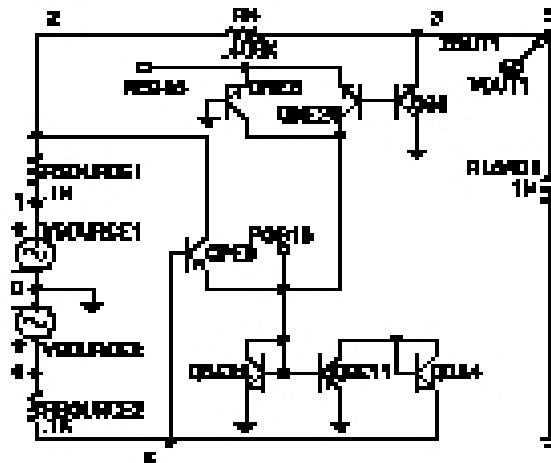


# TIME-OPTIMAL FLY-TO PROBLEM

## CASE 4



# TIME-OPTIMAL FLY-TO PROBLEM – GENERATIONS 0 AND 31 (WITH NEAR- OPTIMAL FITNESS OF 1.541 HOURS)



# **PROMISING GP APPLICATION AREAS**

- **Problem areas where a good approximate solution (but not necessarily optimal solution) is satisfactory**
  - design
  - control
  - data mining
  - forecasting
  - classification
  - image processing and computer vision
- **Problem areas where discovery of the size and shape of the solution is a major part of the problem**
- **Problem areas involving many variables whose inter-relationship is not well understood**

## **PROMISING GP APPLICATION AREAS – CONTINUED**

- **Problem areas where programming by hand is difficult**
  - parallel computers
  - FPGAs
  - cellular automata
  - multi-agent strategies
  - distributed AI
- **Problem areas where large computerized databases are accumulating and computerized techniques are needed to analyze the data**
  - genome and protein sequences
  - astronomy
  - petroleum
  - marketing and financial databases
  - satellite data
  - weather
  - World Wide Web

# **DIRECTIONS FOR FUTURE RESEARCH**

- **Strive for GP results in difficult, real-world problems that typically require human intelligence to solve**
- **Strive for GP results that would be publishable or commercially valuable in their own right**
- **Design of complex structures**
- **Multi-agent strategies**
- **Data-mining**
- **Test script generation / IC testing**
- **Exploiting mental models to solve problems**
- **Auto-parallelization**
- **Grammar induction**
- **Local operation algorithms**
- **Techniques for handling vectors, arrays, and other complex data structures**
- **Assembly code approaches**
- **Evolvable hardware**

- Digital Field Programmable Gate Arrays (FPGA)
- Analog Field Programmable Analog Arrays (FPAA)

# **DIRECTIONS FOR FUTURE RESEARCH – CONTINUED**

- **Modularity**
  - Exploiting many levels of hierarchy
  - Methods for dealing with large numbers of ADFs
  - Finding the right kind of parameterizable modularity for different representations:
    - Circuits, FPGA, Neural Programming
- **Heuristic Exploration**
  - “Creep” operators for constants
  - Interleaved methods of search – crossover, simulated annealing
  - Internal Reinforcement
  - Smart Crossover
  - Use statistics to guide reproductive operations
- **Meta-GP**
  - Automatically tweak the knobs
  - Variable operator percentages
  - Variable population size
- **Issues of Typed GP**

# **DIRECTIONS FOR FUTURE RESEARCH – CONTINUED**

- **Theory**
  - GP Schema theorem
  - What make a problem difficult?
  - Role of deceptiveness in GP
  - Characterizing suitability of representations for problems:
    - Issue of introns
    - Rate of exploration vs. rate of convergence
- **Representations**
  - Dealing with sequences and matrices of data
  - Trees vs. Linear vs. Graphical genomes
  - Phenotype vs. Genotype
  - Handley's statistical computing zones
  - Teller's image processing operators
  - Memory usage

# **DIRECTIONS FOR FUTURE RESEARCH – CONTINUED**

- **Long-term learning and transfer**
  - Goal is to evolve programs for more than a single problem
  - Seeding
  - Noah operator -- seed next run with best of previous
  - Start over with new function set that includes ADFs from previous run
    - Seed the population with either a hand coded or evolved individual by including multiple mutations as seed individuals
    - Learn parameters for the run (such as parameters for the grammar of tree creation) and use from one run to the next

# **DIRECTIONS FOR FUTURE RESEARCH – CONTINUED**

- **Solving multiple problems**
  - Hox gene
  - Use same ADFs for multiple RPBs to solve different problems, or to work together on a single problem
    - Solving multiple problems at the same time might actually be easier, especially if there are possible shared subroutines
- **Learning to learn**
  - Evolving programs that learn
  - Evolving programs that learn and use a complex mental model
  - Evolve a program that performs learning for a neurologically plausible recurrent network of neurons
  - Explore the Baldwin effect with respect to evolving learning

## **7 GP BOOKS AND VIDEOTAPES**

- **Koza, John R. *Genetic Programming: On Programming Computers by Means of Natural Selection*. Cambridge, MA: MIT Press 1992.**
- **Koza, John R. and Rice, James P. *Genetic Programming: The Movie*. Cambridge, MA: MIT Press 1992. (VHS NTSC, PAL, SECAM)**
- **Kinnear, Kenneth E. Jr. (editor). *Advances in Genetic Programming*. Cambridge, MA: MIT Press 1994.**
- **Koza, John R. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: MIT Press 1994.**
- **Koza, John R. *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: MIT Press 1994. (VHS in NTSC, PAL, SECAM)**
- **Peter J. Angeline and Kinnear, Kenneth E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. MIT Press.**
- **Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors).**

***Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University. Cambridge, MA: MIT Press.***

## GENERAL BOOKS ON GENETIC ALGORITHMS

- Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley 1989.
- Holland, John H. *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press 1975. Now available as 2nd edition from The MIT Press 1992.
- Davis, Lawrence (editor). *Genetic Algorithms and Simulated Annealing* London: Pittman 1987.
- Davis, Lawrence. *Handbook of Genetic Algorithms* Van Nostrand Reinhold.1991.
- Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer-Verlag 1992.
- Mitchell, Melanie. 1996. *An Introduction to Genetic Algorithms*. Cambridge, MA: The MIT Press.

## **GP CONFERENCE PROCEEDINGS**

- **Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University.* Cambridge, MA: MIT Press.**
- **Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L. (editors). 1997. *Genetic Programming 1997: Proceedings of the Second Annual Conference, July 13-16, 1997, Stanford University.* San Francisco, CA: Morgan Kaufmann.**

## **1997 CONFERENCES**

- **FEA-97 – March 3 – 5, 1997 – Research Triangle Park, NC**
- **GP-97 – Stanford, July 13-16 (Su-W), 1997**  
E-MAIL: `gp@aaai.org`  
WWW: `www.genetic-programming.org`
- **ICGA-97 – July 20 – 23, 1997 – East Lansing, Michigan**
- **AAAI-97 – Providence, RI – July 27 – 31, 1997**
- **ECAL-97 - July 28 – 31, 1997, Brighton, UK**
- **IJCAI-97–Nagoya, Japan – August 24-29, 1997**
- **GALESIA-97, September 1997 - Glasgow**

## **1998 CONFERENCES**

- **EP-98 –March 25-27 (W-F), 1998 – San Diego**
- **ALIFE-98**
- **ICEC-98 – May 1998 – Anchorage, Alaska**
- **FOGA-98 – Charlottesville, VA – July 18 – 21, 1998 (TENTATIVE!!!)**
- **GP-98 – Madison, Wisconsin – July 22-25 (W-Sa), 1998**

E-MAIL: [gp@aaai.org](mailto:gp@aaai.org)

WWW: [www.genetic-programming.org](http://www.genetic-programming.org)

- **AAAI-98 – Madison, Wisconsin – July 26 - 30, 1998 (Su-Th)**
- **ICES-98 - Lausaune - Sept 24 - 26 (Th-Sa), 1998**
- **PPSN-98 – Amsterdam - Sept 27 - Oct 1 (Su-Th), 1998**

## **1999 CONFERENCES**

- **IJCAI-99 – Stockholm – July 31 - Aug 6, 1999**

## **GP E-MAIL LISTS**

### **GENETIC PROGRAMMING (GP) LIST**

- To subscribe, send e-mail message to:  
**Genetic-Programming-Request@CS.Stanford.Edu**
- Be sure to send to exactly this address, (which includes the word "Request")!
- The BODY of your message must consist of exactly the words:  
**subscribe genetic-programming**

### **SPECIAL BAY AREA GP LIST**

**(USED ONLY TO ANNOUNCE LOCAL GP MEETINGS AND BAY AREA GP LUNCHES)**

- To subscribe, send e-mail message to:  
**BA-GP-Request@CS.Stanford.Edu**
- Be sure to send to exactly this address, (which includes the word "Request")!
- The BODY of your message must consist of exactly the words:  
**subscribe BA-GP**

# JOHN KOZA'S HOME PAGE

**`http://www-cs-faculty.stanford.edu/~koza/`**

## **Contains**

- Information on GP-96, GP-97, GP-98 conferences
- Links to people doing GP research
- List of PhD theses in progress
- Links to many other GP resources
- Abstracts of JK's publications
- Links to other GP WWW pages
- Links to Langdon's complete GP bibliography